VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Čínská dáma

Chinese checkers

2019 Lucie Kozarová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **Lucie Kozarová**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Čínská dáma
Chinese Checkers

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je implementovat počítačovou podobu deskové hry Čínská dáma.

1. Popište zadanou deskovou hru a její pravidla.
2. Proveďte objektově orientovanou analýzu zvolené varianty s využitím návrhových vzorů.
3. Hru implementujte (hra člověk-počítač).
4. Sehraje několik ukázkových partií a výsledky vyhodnoťte.

Seznam doporučené odborné literatury:

[1] Arnold, Peter: The Complete Book of Indoor Games.
[2] Zapletal, Miloš: Velká encyklopedie her - Hry v klubovně. Olympia, Praha, 1986.
[3] http://www.deskovehry.info/pravidla/cin-dama.htm
Dále podle pokynů vedoucího práce

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Jiří Dvorský, Ph.D.**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2019

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

prof. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 30. dubna 2019 .....

Ráda bych poděkovala vedoucímu práce doc. Mgr. Jiřímu Dvorskému, Ph.D. za pomoc s tvorbou této práce. Dále také své rodině za jejich velkou podporu.

Abstrakt

Tato bakalářská práce se zabývá tvorbou hry zvané Čínská dáma. Cílem této práce je za pomoci objektově orientované analýzy vytvořit funkční verzi hry. Program je vytvořen pro hru dvou hráčů, v tomto případě člověk proti počítači. Projekt je realizován v programovacím jazyce Java. Součástí práce je také popis a historie deskové hry Čínská dáma, která je zde převedena do počítačové podoby. Výsledkem práce je funkční provedení hry i s ukázkovými partiemi.

Klíčová slova: Čínská dáma, Java, objektově orientované programování, desková hra

Abstract

This bachelor thesis is aimed on programming a game called Chinase checkers. The goal of the thesis is to create a functional version of the game by using object-oriented analysis. The program is designed for two players in this case a human versus computer. The project is realised in Java language. The thesis also contains description and history of a board version of Chinase checkers which is converted into a computer game. As a result the thesis provides a functional implementation of the game together with demonstration of examples of some matches.

Key Words: Chinese checkers, Java, Object-oriented programming, board game

Obsah

Seznam použitých zkratek a symbolů

MVC	–	Model View Controller
px	–	pixel
GUI	–	Graphical User Interface
AI	–	Artificial Intelligence

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Tato práce se zabývá hrou zvanou Čínská dáma. Jedná se o poměrně starou deskovou tahovou hru, která však v našich končinách není příliš známá. Je však do jisté míry podobná klasické dámě, se kterou sdílí i část svého názvu. Práce tak bude obsahovat popis rozdílů těchto dvou her, aby Čínskou dámu čtenářům více přiblížil. Jednou z podobností obou verzí dam je, že pro oboje existuje více různých sad pravidel. Tímto se tak odlišují od další světoznámé deskové hry, a to šachů, které mají pravidla pouze jedny. Na rozdíl od obou těchto her zde nemusí soupeřit jen dva hráči, ale může jich být až šest.

Cílem práce bude vytvořit funkční program pro hru jeden na jednoho, v tomto případě půjde o partii mezi lidským hráčem a umělou inteligencí (jinak také nazývanou jako AI). Celý program bude vytvářen za pomoci objektově orientovaného návrhu v programovacím jazyce Java. Tvorbě aplikace však musí předcházet analýza a návrh, stejně jako bude nutné specifikovat pravidla, kterými se hra bude řídit. Analýza bude spočívat v rozboru některého z již existujících řešení.

Na základě poznatků, které při analýze získáme, vznikne prvotní návrh samotného řešení. Jeho postupným vylepšováním by se aplikace měla stát konkurenceschopnou s původně analyzovaným řešením. Většina autorů her však neuvádí, jaké algoritmy pro svoji umělou inteligenci používají. Vytvoření umělé inteligence, která původní řešení předčí, se tak nemusí podařit. Součástí práce tak bude i následný rozbor partie, ze kterého by mělo vyplynout, z jakého důvodu námi vytvořený algoritmus umělé inteligence vyhrál, či naopak.

2 Čínská dáma

2.1 Původ hry

Čínská dáma vznikla již v devatenáctém století. I přes svůj název však vznik této hry nepochází z Číny, ale z Německa. Je to upravená verze Halmy, která se hrála na čtvercovém herním poli. U Halmy měli hráči své figurky rozestavěny na protějších rozích čtverce a cílem bylo dostat všechny své figurky do protějšího rohu. Herní pole Čínské dámy je upraveno do tvaru šesticípé hvězdy, ale princip zůstává stejný, dostat své figurky na protější stranu (tedy protější cíp hvězdy). Původní název hry zněl "Stern-halma". "Stern" v němčině znamená hvězda. Když se hra dostala i do Spojených států amerických, vzniklo pro ni jméno "Hop Ching Checkers". Pozdější název "Čínská" dáma vznikl pravděpodobně z důvodu tehdejšího zájmu Ameriky o věci z orientu. [8]

2.2 Popis hry

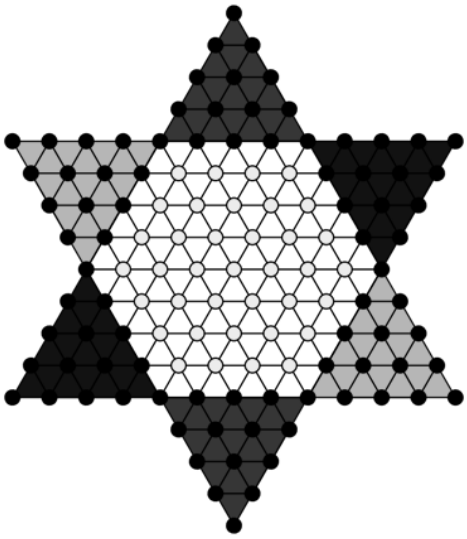
Čínská dáma patří mezi deskové strategické hry. Počet hráčů může být dva, tři, čtyři nebo šest. Herní pole je ve tvaru šesticípé hvězdy. Každý cíp hvězdy je tvořen trojúhelníkem a vnitřní pole má tvar šestiúhelníku. Prohlédnout si jej můžete na obrázku číslo 1. [3]

Každý hráč má sadu deseti figurek, pouze při hře dvou hráčů se jich může používat patnáct. Rozestavení na hvězdici je pro dva a čtyři hráče zvoleno tak, aby byli vždy na dvou protějších cípech hvězdy, na obrázku číslo 1 jsou tyto cípy hvězdy zobrazeny stejnou barvou. Při třech hráčích se nechává mezi každým z nich jeden volný cíp. Nejtypičtější rozložení je takové, že se zaplní horní cíp hvězdy a dva dolní cípy hvězdy, které jsou na krajích. U šesti hráčů se zaplní všechny cípy hvězdy. [5]

Cílem hry je dostat své figurky do protějšího cípu hvězdy, který je označován pojmem domeček. Hráč může ve svém tahu posunout figurku o jedno pole buď v diagonálním směru, nebo do boku, případně může provést skok přes sousední figurku. Vítězem hry se stává ten, kdo jako první dostane všechny své figurky do domečku. Více o možných pohybech figurek se dozvíte v kapitole věnované pravidlům hry. [2] [4] [6]

2.3 Pravidla hry

Hráči se na tahu střídají ve směru hodinových ručiček. Není pevně stanoveno, kdo začíná hru. V případě hry člověka proti počítači je typické, že začíná člověk. Táhnout figurkami lze hned několika způsoby. Při klasickém posunu lze figurku přesunout buď po diagonále, či do boku na volné sousední pole, viz obrázek č. 2. Stejně jako v klasické dámě, i v této podobě hry existuje možnost přeskočit figurku, příklad takového tahu je zobrazen na obrázku č. 3. Rozdílem však je, že nelze přeskakovat pouze soupeřovy figurky, ale také vlastní. Navíc, při přeskočení figurky nedojde k jejímu odstranění z herního pole. Skok lze provést pouze v přímém směru, přes sousední obsazené pole, pokud je volné místo za ním. Zřetězením skoků vzniká cesta, což je jediný způsob, kterým v této hře lze provést více tahů v jednom kole (ty jsou povoleny pouze



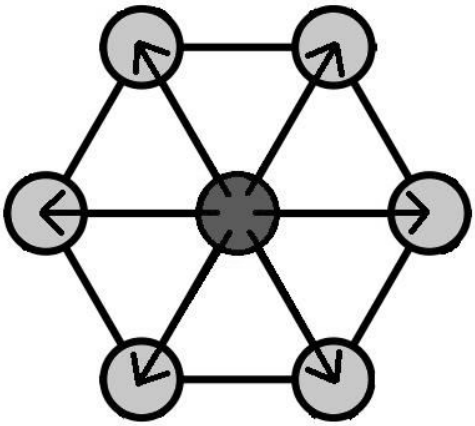
Obrázek 1: Herní plocha Čínské dámy [3]

pro figurku, se kterou byl proveden první skok). Směr skoku se může po každém tahu změnit. Ukázku cesty můžete vidět na obrázku č. 4. Skok však nelze kombinovat s posunem o jedno pole. Celkový tah je také zakázáno ukončit v jiném cípu hvězdy, než který patří hráči, nebo jeho cíli (domečku). Některá pravidla také říkají, že pokud již hráčova figurka vstoupí do domečku, nemůže ho již opustit. Může se však pohybovat a také přeskakovat jiné figurky, které se nacházejí v tomto cílovém poli. [1] [4] [6]

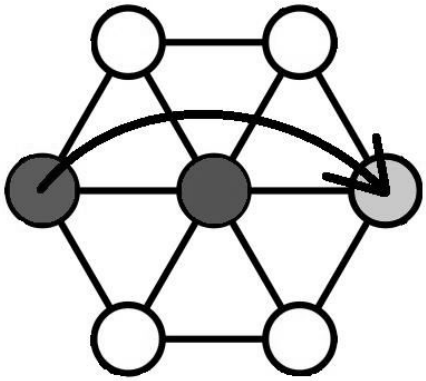
2.4 Rozdíly oproti klasické dámě

Rozdíly proti tahům v klasické dámě lze tak shrnout na tyto:

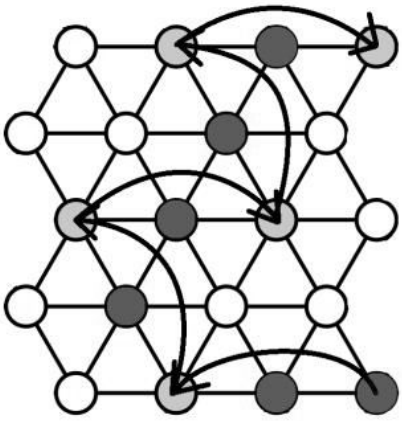
1. Figurkami lze pohybovat nejen po diagonálách, ale také do bočních pozic.
2. Při přeskočení figurky nedochází k jejímu odstranění z pole.
3. V obou variantách dámy lze přeskakovat i více figurek, je-li za každou volné pole, na kterém by figurka mohla ukončit tah. Oproti některým variantám klasické dámy hráč není povinen soupeřovy figurky přeskakovat.
4. Figurkami není nutné posunovat směrem k cílovému poli, lze přeskakovat i posouvat směrem dozadu, k hráčovu původnímu poli.
5. Žádná figurka hráče nesmí na konci tahu skončit v jiném cípu hvězdy, než v hráčově původním nebo cílovém.
6. Pokud figurka ukončí tah v cílovém poli, nezmění se její podoba (např. na dámu), ale toto cílové pole již nemůže opustit.



Obrázek 2: Možné tahy při posunu o jedno pole



Obrázek 3: Skok přes figurku



Obrázek 4: Zřetězení skoků

7. Cílem hry není postupně přeskočit a tím odstranit všechny soupeřovy figurky, ale dostat všechny své figurky do domečku dříve, než tak učiní jiný hráč. [5]

2.5 Jiné varianty hry

U některých her jsou povoleny pouze tahy, které hráče posouvají vpřed, stejně jako u klasické dámy. Z toho plyne, že tahy vzad jsou zakázány.

Můžeme se také setkat s případem, ve kterém je povoleno figurkou zůstat v cípu hvězdy, která není od daného hráče, ani jeho cíle.

Existuje také varianta čínské dámy, kde jsou umožněny skoky o víc, než jen o dvě pole. Přeskakovaná figurka totiž nemusí být sousední, ale musí být vždy ve středu skoku, což znamená, že před i za ní musí být stejný počet volných políček.

Hra pro dva, případně pro tři hráče, se dá pozměnit i tím, že každý hráč bude ovládat více sad figurek najednou. Při hře dvou lidských protivníků se tak na herní ploše může objevit i všech šest sad figurek, kde každý z hráčů ovládá jejich polovinu. Zachovává se pravidlo, že sady figurek jsou vždy v takovém počátečním rozestavení, aby v protějších cípech byly vždy nepřátelské figurky. Hráč se musí pak dostat do cíle s každou svou sadou figurek. [4] [6]

3 Analýza hry

Pro pochopení principu hry a hlavně chování počítače bude nejprve odehráno několik partií Čínské dámy. Pro tyto účely se využijí některé již vytvořené hry, které můžeme nalézt na internetu. Pro analýzu byla zvolena verze hry, která neumožňuje dlouhé skoky. [9] [10]

Cílem hry je dostat všechny své figurky do protějšího cípu hvězdy dříve, než soupeř. Na následujících řádcích budou rozebrány typické rysy chování počítačového hráče, které se daly vypozorovat z odehraných her. Podstatnou informací, která se dá tímto pozorováním určit je, že se počítač v každém svém tahu posunul blíže ke svému domečku, nikdy neprovedl tah směrem dozadu. Další patrnou věcí je, že jestliže má počítač možnost provést skok, či lépe zřetězení skoků, tak tuto možnost téměř vždy využije, pokud se tímto způsobem posune směrem dopředu. Prakticky nenastane situace, kdy by počítač posunul svou figurku jen doprava, či doleva na řádku, ani když by mohl přeskočit nějakou jinou figurku na daném řádku a posunout se tak o dvě políčka. Takový posun by jej k cíli nepřiblížil. Tah pouze o jedno políčko počítač využije až v případě, kdy nemá žádnou možnost skoku směrem dopředu. [9]

Toto budou základní úvahy, podle kterých budou vytvářeny funkce pro správné chování počítačového hráče. Na základě těchto úvah se vytvoří priority, podle kterých bude počítačový hráč volit svůj pokud možno nejlepší tah. Jak již bylo řečeno, pokud počítač může provést skok, téměř vždy této možnosti využije. Proto hlavní pravidlo bude pro vyhledávání skoků. K druhému pravidlu přejdeme teprve ve chvíli, kdy nebude mít žádná figurka počítače možnost skoku. V tomto případě bude počítač hledat možnosti posunu pouze o jedno políčko. Tato dvě hlavní pravidla postačí pro základní implementaci vyhledávání možných tahů. [9]

Dále budou v textu blíže rozepsány způsoby, jakým počítačový hráč vybírá svůj tah v odehraných partiích. Po odehrání několika her bylo možno vypozorovat, že první dva tahy jsou u počítačového hráče vždy stejné, případně jen zrcadlově odlišné. Jako první tah si vždy zvolí krajní figurku na třetím řádku. S figurkou vlevo na třetím řádku skočí doprava dolů a tím svůj tah ukončí. V následujícím tahu může provést rovnou dva skoky, směrem doleva dolů, s figurkou z prvního řádku. Když si počítač pro první tah vybere pravou figurku, tak s ní provede skok doleva dolů a následně opět s figurkou z prvního řádku provede dva skoky směrem doprava dolů. Následující tahy již nejsou přesně dány, ale dá se vypozorovat, že si počítač vybírá řešení taková, která mu umožní vytvoření lepší cesty pro příští tahy. Toto chování lze pozorovat již na uvedených prvních dvou tazích. Na začátku hry může provést skok se kteroukoli figurkou ze třetího řádku, a to do obou dolních směrů. Přesto si vždy vybírá pouze krajní a vždy s touto figurkou provede tah ve směru, který zde byl uveden. Byla by ještě možnost, aby v prvním tahu figurka skočila do druhého směru a následně figurka z prvního řádku by v druhém skoku také musela skočit v druhém směru, než jsme si uvedli. Výsledné rozestavění figurek počítače po prvních dvou tazích by i tak bylo opět stejné. Skok s jednou z uvedených krajních figurek v prvním tahu je jediná možnost, kterou si v následujícím tahu může počítač umožnit skočit rovnou dvakrát s figurkou z prvního řádku a přiblížit se tak s ní rovnou o čtyři políčka k domečku. [9]

Pokud by mělo být docíleno takovéto chování v naší implementaci, musela by být vytvořena metoda, ve které by se vyhledávaly všechny možné tahy, a na každý z těchto tahů by tento výpočet probíhal znovu. Podle toho kolik tahů dopředu by se takto chtělo vyhledávat, tolikrát by muselo proběhnout zanoření v rekurzi pro vyhledávání možných tahů.

Sledováním chování počítačového hráče se dá zpozorovat ještě jedna, ne příliš patrná situace. A to, že si počítač své tahy v případě pouhého posunu o jedno pole volí tak, aby nevytvořil možnost skoku pro svého soupeře. Při tazích, kde provádí skok, se toto chování příliš pozorovat nedá. [9]

3.1 Návrh programu

Předchozí kapitoly se věnovaly popisu Čínské dámy. Proběhlo seznámení se vzhledem hry, jejími pravidly a rozbor průběhu odehraných partií. Další kapitoly se již budou zabývat samotnou tvorbou programu.

Program bude vytvářen v programovacím jazyce Java. Tento jazyk je sám o sobě objektově orientovaný, proto je pro tvorbu této práce vhodný.

Nejprve se vytvoří návrh rozvržení celého projektu do jednotlivých tříd.

Čínská dáma vznikla původně jakožto desková hra, ve které se hrálo s figurkami na herní ploše ve tvaru hvězdy. Toto bude potřeba přenést do počítačové podoby. Návrh musí být vytvořen tak, aby s ním mohl pracovat počítač. [4]

Jedna z tříd bude muset reprezentovat samotnou herní plochu. Jelikož se jedná o základní stavební prvek celé aplikace, který obsahuje mnoho informací, bude již na začátku dobré počítat s budoucím rozdělím této třídy. Toto rozdělení bude nutné zejména kvůli přehlednosti celého projektu. Již na začátku se dá uvažovat s rozdělením této plochy na tři části. A to na část, která bude uchovávat informace o samotných políčkách a pozicích figurek na této herní ploše (tzv. datovou vrstvu aplikace). Druhá část se bude věnovat samotnému vykreslování této plochy (tzv. zobrazovací vrstvu programu). Takto by však zůstala aplikace pořád ve stejném stavu, je tedy nutné zavést ještě třetí část, která se bude věnovat obsluze akcí uživatele (tzv. řídící část aplikace).

Z tohoto návrhu tedy prozatím vyplývá, že bude zapotřebí zavést entity herní plocha, políčko a figurka. Vzhledem k tomu, že se políček a figurek v této hře vyskytuje více, bude pravděpodobně nutné, seskupit je v logických celcích, například v lineárních seznamech. Tyto entity by pro jednoduchost stačily k obsluze a ukládání informací pro tah lidského hráče. Součástí aplikace je však i jeho protivník, počítačový hráč. Ten bude také využívat stejného datového návrhu herní plochy, ale již nebude přímo závislý na zobrazovací vrstvě aplikace.

Počítačový hráč bude potřebovat další vlastní třídy, které by mu pomohly s výběrem jeho nejlepšího tahu. Jak již bylo řečeno v předešlých kapitolách, jeho nejlepší tah je obvykle ten, který jeho figurku co možná nejblíže přiblíží cílovému poli. Toho jde zpravidla dosáhnout pomocí skoků. Z odehraných her je také známo, že pokud pro počítač existují stejně dobré tahy, vybere si z nich jeden náhodně. Toto bylo patrné již v prvním tahu počítače, kdy vždy vybral ke skoku

figurku nacházející se v třetí řadě. Tah však provedl různým směrem – do cíle ho všechny tyto tahy přiblížily stejně. [9]

Z tohoto jednoduchého návrhu lze usuzovat, že pro implementaci počítače budou zapotřebí pomocné třídy, které umožní pro každou figurku ukládat jednotlivá řešení zvlášť. Každé takovéto řešení bude obsahovat seznam políček, na která se může daná figurka počítače dostat. Pomocí metody, která bude obstarávat výběr z možných tahů, se bude vybírat to nejvhodnější řešení, případně několik nejvhodnějších.

Tím nastane situace, kdy počítač bude mít jednotlivá řešení pro každou figurku samostatně (každá figurka bude obsahovat seznam pouze svých řešení). Již z prvního tahu lze zpozorovat, že figurky ze čtvrté řady nemají tak dobré řešení (mohou se jen pohybovat o jedno pole vpřed), jako ty ze třetí (mohou přeskočit figurku ze čtvrté řady). V návrhu tak bude muset existovat funkce, která dokáže probrat jednotlivá řešení figurek a vybrat z ní podmnožinu těch nejlepších. Z těchto si již lze vybrat finální tah, který je pro počítačového hráče nejvýhodnější.

Pro návrh počítače se dá již v tuto chvíli uvažovat o třídách týkajících se seznamu figurek a jejich tahů. Každá figurka bude obsahovat seznam řešení (týkajících se pouze této figurky). Toto řešení se bude skládat z posloupnosti tahů. Každý tah se bude obsahovat souřadnice políček, přes která musí figurka projít.

3.2 MVC

Jak již bylo řečeno, program bude rozdělen do tří hlavních logických části. V této kapitole se budou blíže vysvětlovat důvody pro toto rozdělení. Projekt budeme vytvářet na základě architektury MVC. Význam této zkratky je Model View Controller. Jednou z výhod, kterou tento architektonický vzor přináší, je rozdělení zdrojového kódu do logických celků. Tím také dochází ke značnému zpřehlednění jednotlivých tříd. Použití MVC se vyplácí obzvláště ve velkých projektech. Je snazší provádět úpravy v kódu, když jsou jednotlivé části logicky strukturované. Přidání nových funkcionalit do aplikace se pak stává mnohem jednoduší. Stejně tak se v takovém kódu dají lépe nalézt případné chyby a provést jejich opravu. [11] [12]

Postupně si popíšeme všechny tři části. Začneme modelovou částí.

Jako Model se zde označují části kódu, které mají na starost datovou strukturu programu. Dále sem patří také takzvaná business logika, někdy zvaná jako doménová logika. [11] [12]

Controller zde zastává funkci komunikátoru mezi Modelem a View. Obstarává samotnou funkcionalitu programu. Na základě dat, která jsou mu zasílána, určuje, co se s nimi má stát.[11] [12]

Poslední částí je pak vrstva zobrazení - spíše známá pod anglickým názvem View. Tento View má za úkol zprostředkovat grafické zobrazení jednotlivých modelů. Je to přesně tato vrstva, kterou jako jedinou uvidí běžný uživatel aplikace pouhým okem. Z toho také plyne, že jakákoliv akce, kterou uživatel provede, je nejprve zachycena právě v této části aplikace. Proto musí přímo spolupracovat s vrstvou řízení - controllerem, kterému předává jednotlivé podněty od uživatele.

Těmi může být nejen zachycení kliku myši, ale také stisknutí určité klávesy, či jejich kombinace. O významu této zachycené akce však rozhoduje zpravidla controller.[11] [12]

Nejčastěji se MVC používá při tvorbě webových aplikací. Projekt, který bude vytvářen v této práci, není typický pro použití této architektury, přesto si z ní můžeme vzít příklad toho, jak mít co nejlépe uspořádané rozdělení projektu do jednotlivých částí.

Projekt může obsahovat samozřejmě více modelů, controllerů i viewů. [11] [12]

4 Tvorba počítačového hráče

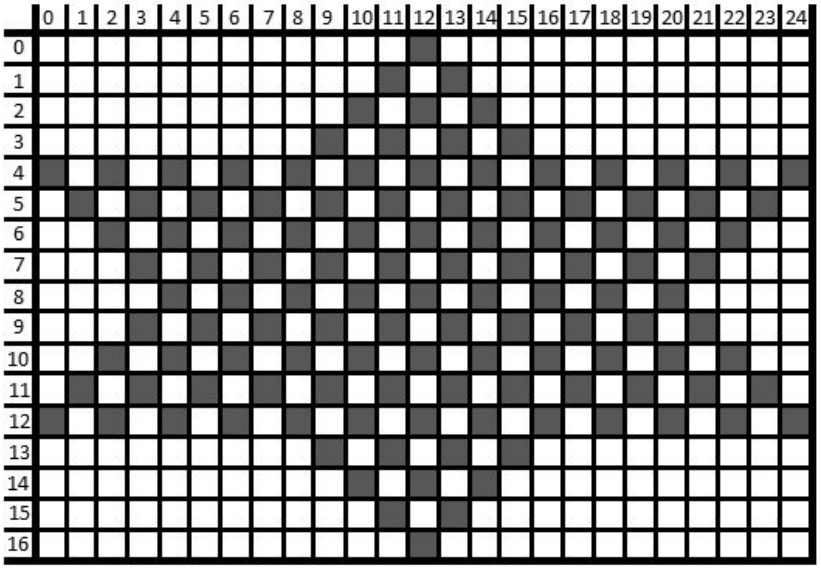
4.1 Herní pole

Základním prvkem aplikace je samotná hrací plocha, dále pojmenovaná jako herní pole. Jak již bylo řečeno v popisu hry, herní pole má tvar šesticípé hvězdy, tudíž nemá tvar klasické šachovnice. Políčka ani nejsou umístěna pod sebou v jednotlivých sloupcích. Při průzkumu jejího tvaru je však patrné, že pozice políček na sudých řádcích jsou od kraje stejně vzdálené. Při vypuštění lichých řádků by tedy plocha již začínala připomínat tvar obdélníku. Stejný princip však platí také obráceně, a to při vynechání sudých řádků. Jednotlivé řádky se ale i tak liší v počtu políček, která obsahují. Základní úvahou tak může být využití fixního dvojrozměrného pole. Při jeho grafickém vyobrazení by stačilo liché řádky předsunout. Toto řešení však špatně reaguje na jiný počet políček na každém řádku. Hledání sousedních polí by tak muselo být ošetřeno spoustou podmínek. [4]

Pro rychlost výpočtu umělé inteligence je procházení skrz jednotlivá políčka pole stěžejní. Nabízí se tedy uložení políček do některé struktury na bázi stromu. Tyto struktury umožňují rychlé procházení po jednotlivých sousedících políčkách. Problém však nastane, že tento strom by neobsahoval číselné hodnoty, ve kterých se dá cílová hodnota rychle nalézt. Při tahu hráče, který provádí kliknutím myši do vyobrazené herní plochy, by se musela projít značná část tohoto stromu, než by se konkrétní políčko nalezlo. Také by se musela navrhnout funkce, která by procházela daným stromem a přizpůsobila by ho k vykreslení. Průchod dvojrozměrným polem je pro vykreslení značně jednodušší. V tomto kroku tedy jsou v úvaze dvě struktury, kde jedna umožňuje rychlé vyhledání konkrétních políček (struktura na bázi pole) a druhá jednoduchý a rychlý průchod (struktura na bázi stromu).

Pro tuto práci tedy bude zvolena struktura, která by umožnila výhody obou těchto struktur zkombinovat.

Klasická herní plocha v této práci obsahuje přesně sedmnáct řádků a třináct sloupců. Definovaná struktura bude na bázi dvourozměrného pole. Musí tedy obsahovat stejný počet řádků (sedmnáct) jako klasická plocha, ale sloupců bude obsahovat téměř dvojnásobné množství, dvacet pět. Poslední sloupec je nutné přidat z důvodu nerovnoměrného odsazení lichých a sudých řádků. Mezi jednotlivá políčka tohoto pole budou vložena prázdná místa. Tato struktura i vizuálně odráží vzhled reálného pole. Struktura je tak vhodná nejen pro výpočet souřadnic políčka, na které uživatel poklepal myší, ale i pro průchod mezi políčky. Jelikož je struktura obdobná reálnému vyobrazení, tak její vykreslení nebude potřebovat zvlášť složitou funkci. Nevýhodou této struktury oproti výše uvedeným je paměťová náročnost. Poměrně velké množství políček je v ní totiž neobsazené. Aplikace využívá jen jednu herní plochu, čili se tato struktura v průběhu hry nikdy duplikovat nebude. Vyšší paměťové nároky tak nejsou překážkou. Dopočítání indexů jednotlivých sloupců lze provést pomocí dvou údajů, a to počtu políček řádku (fixní údaj) a znalosti, zdali se funkce nachází na sudém, či lichém řádku. Při vykreslování tedy také nebude nutné



Obrázek 5: Herní pole

procházet celé toto pole včetně prázdných míst.

Na obrázku č. 5 je takto definované pole znázorněno. Prázdná políčka na obrázku jsou neplatná, zvýrazněná pak znázorňují součást herní plochy.

Pro lepší přehlednost při práci s tímto herním polem bude využit výčtový typ hodnot, kterým bude toto pole naplněno. Pro zadefinování všech stavů jednotlivých políček bude potřeba minimálně pět různých hodnot. Jmenovitě jde o tyto hodnoty:

- *neplatné políčko,*
- *volné políčko,*
- *figurka hráče,*
- *figurka počítače,*
- *okrajové cípy.*

Neplatnými políčky rozumíme všechny takové, které jsou umístěny mimo hrací plochu (šesticípou hvězdu). Na obrázku číslo 5 byly zobrazeny prázdnými čtverečky. Volné políčko je takové, které je již součástí šesticípé hvězdy a lze na něj umístit figurku, momentálně na něm žádná nestojí. Hodnoty "figurka hráče", resp. "figurka počítače" tedy naopak označují volné políčko, na kterém je figurka od příslušeného hráče umístěna. Aby byla šesticípá hvězda kompletní, potřebujeme ještě jednu hodnotu v tomto výčtovém typu. Tato hodnota bude označovat všechny cílová pole nehrajících hráčů, do kterých nesmí zajít ani figurka hráče, ani počítače.

Případů, kdy by nastala situace zřetězení skoků tak, aby se využila možnost skoku přes některý z okrajových cípů hvězdy, je pouze zanedbatelné množství. Pro průběh hry nebude mít žádný vliv, když tato možnost nebude vůbec povolena.

Herní plocha na začátku hry vypadá vždy stejně, proto bude celé pole vyplněno již zmíněnými hodnotami fixně. V této struktuře se budeme pohybovat dvojím způsobem. Při výpočtu tahů počítače vždy od předem známých pozic figurek k jejím sousedům. U tahů lidského hráče bude existovat funkce, která přepočítá souřadnice kliknutí v pixelech (px) na indexy tohoto pole.

4.2 Zjišťování sousedních polí

V předchozím bodě bylo vytvořeno dvourozměrné pole, které bude reprezentovat aktuální stav herního pole.

Tato třída však nemá za úkol data pouze uchovávat, ale také k nim umožnit ucelený přístup pro ostatní třídy.

Pro pohyb jednotlivých figurek bude nejprve potřeba určit, jakou množinu sousedů obsahují daná políčka. Pro výpočet skoků bude potřeba přidat další metody, které umožní zjištění, zdali je sousedem nejprve figurka, a zda se až za ní nachází volné políčko. Jelikož se figurka může vždy v jednom tahu pohybovat pouze určitým směrem, tak tyto metody mohou být modifikovány tak, aby nevyhledávaly všechny sousední políčka, ale jen ta ze zadaného směru.

V poli je umožněno pohybovat se do stran na stejném řádku a v diagonálních směrech.

Možnosti směrů jsou tyto:

- *doleva,*
- *doprava,*
- *doprava nahoru,*
- *doleva nahoru,*
- *doprava dolů,*
- *doleva dolů.*

Každé políčko má své souřadnice, podle kterých se vyhledávají jeho sousední políčka. Souřadnice jsou indexy řádku a sloupce v dvourozměrném poli.

Sousední políčka v diagonálních směrech budou vypočítána podle již zmíněných souřadnic. Ve směru nahoru se od indexu řádku odečítá jednička, ve směru dolů se naopak přičítá. Stejně tak index sloupce se mění podle toho, jestli se posouváme po diagonále doleva, či doprava. Doleva odečítáme jedničku od souřadnice sloupce, doprava tedy jedničku přičítáme. Při posunech do stran se musí brát v úvahu neplatná políčka, proto je zde posun ne o jeden index, ale rovnou o dva. Směr doleva tedy odečítá od souřadnice sloupce dvojku, doprava pak naopak přičítá dvojku.

Tabulka 1: Zobrazení výpočtu indexu pro sousední políčka

Směr	Souřadnice souseda
Doleva	[řádek][sloupec-2]
Doprava	[řádek][sloupec+2]
Doprava nahoru	[řádek-1][sloupec+1]
Doleva nahoru	[řádek-1][sloupec-1]
Doprava dolů	[řádek+1][sloupec+1]
Doleva dolů	[řádek+1][sloupec-1]

V tabulce č. 1 jsou tyto souřadnice přehledně rozepsány. U každého směru bude přidána kontrola indexu případného souseda. Tato kontrola bude zamezovat situacím, kdy by zadané souřadnice souseda ležely mimo rozměry pole. Pole má neměnnou velikost, proto lze do kontrolních podmínek zadat známou velikost indexu, za kterým nemá smysl hledat sousední políčko. Tímto se zamezí neustálému opakování dotazu na rozměry herního pole. Indexy řádků musí být v rozmezí nula až šestnáct a indexy sloupce nula až dvacet čtyři. V podmínkách pak stačí zadávat dotazy pouze pro daný směr, ve kterém se nachází hledané sousední políčko.

4.3 Vyhledávání tahů

Z výše uvedených metod již lze procházet pole pomocí sousedících políček. Následuje tak úvaha o problému samotného pohybu počítače. Umělá inteligence má za úkol vybrat nejvhodnější tah. V prvním kroku je tak nutné nejdříve nalézt všechny možné tahy, které lze s jeho figurkami provést.

Princip vyhledání těchto tahů může být různý. První možností je, že pro každou figurku algoritmus vyhledá nejkratší možnou cestu k cíli. Tato metoda by pracovala s dynamickým seznamem tahů, které musí každá figurka vykonat. Toto řešení má však několik úskalí. Mohou nastat situace, kdy kvůli postavení soupeřových figurek nebude existovat cesta do cílového pole. Příkladem by mohla být situace, kdy by v prvním tahu začínal hru počítač. Jelikož má soupeř ještě všechny své figurky na startu, obsazuje tak všechny cílové pole počítače sám. Pokud by začínal tah hráč, jak to bývá zvykem, bude se algoritmus v prvním kroku snažit všechny své figurky dopravit nejkratší cestou do právě odkrytého pole hráčem. V průběhu hry také může dojít k situaci, kdy by figurka mohla přeskočit hned několik soupeřových kamenů, ale dostat se tím do situace, odkud cesta k cíli nevede. Algoritmus by tak výhodné zřetězení skoku nevykonal, jelikož by po něm nenašel vhodnou cestu k cíli. Naopak by za kratší cestu mohl prohlásit posun po jednom políčku okolo soupeřových figurek. Problém toho řešení také spočívá v určení, s jakou figurkou je nejvýhodnější provést tah. Vybrání figurky, pro kterou algoritmus našel nejkratší cestu k cíli, by vedlo k řešení, kdy je opakovaně vybírána stejná figurka. Je totiž jediná, která se k cíli přibližuje a tím svoji cestu zkracuje.

Druhou možností vyhledání tahů je princip, při kterém počítač nalezne všechny tahy, které může v daném kole zahrát se svými figurkami. Nevybírá tak posloupnost svých tahů. Tyto jednotlivá řešení si opět uchová například v dynamickém seznamu. V daném tahu tak vznikla počítači množina řešení, ze které musí zvolit jedno pro něj nejvýhodnější. Aby mohl rozhodnutí provést, musí nejprve všechna řešení ohodnotit. Je však pravděpodobné, že za nejvýhodnější nebude označeno jen jedno řešení, ale hned několik. Toto chování je pozorovatelné hned při prvním tahu, kdy existuje hned několik stejně výhodných skoků. Algoritmus tak z celkové množiny řešení vybere podmnožinu, ze které již tah může vybrat za pomoci náhodného výběru.

Na rozdíl od první varianty, lze tato druhá široce modifikovat. Pro zjednodušení výpočtu lze již v prvním kroku ukládat pouze řešení, která figurky přiblíží k cíli, jde tedy o posun, či skok směrem dolů. Pokud je však po projití všech figurek množina prázdná, lze přistoupit k jejímu rozšíření, tím by bylo vyhledání tahů, které umožňují pohyb do stran. Obdobně se dá využít možnosti nejprve vyhledat pouze skoky, a až v případě prázdné množiny vyzkoušet posuny jak směrem dolů, tak směrem do stran. Toto řešení by také umožnilo zapojit predikci, kdy by algoritmus vyzkoušel několik tahů dopředu, poté situaci na herní ploše vyhodnotil, a tím ovlivnil hodnocení v prvotním tahu.

Využití druhé varianty se tedy zdá výhodnějším. Algoritmus tedy bude muset obsahovat následující kroky:

1. nalezení množiny řešení,
2. ohodnocení množiny řešení,
3. výběr podmnožiny řešení na základě ohodnocení,
4. výběr a následné provedení jednoho konkrétního řešení z vybrané podmnožiny.

Všechny tyto kroky lze nezávisle modifikovat a každý z nich má přímý vliv na konečný výběr tahu. Například vynecháním všech skoků by se docílilo toho, že bude algoritmus celou hru pouze figurkami posouvat o jedno políčko. Stěžejní část výběru se však skrývá v druhém a třetím bodě. Zvolení optimální hodnotící funkce bude předmětem dalších kapitol. Třetí krok, a to výběr z podmnožiny řešení, obvykle vybírá jen řešení stejné, a to nejvyšší hodnoty. Při použití striktní hodnotící funkce můžeme definovat toleranci, která se má při výběru uplatnit (pokud to bude nutné pro lepší výběr tahů). Při situaci, kdy jednotlivé řešení mají ohodnocení: A = šest, B = pět a C = dva. S tolerancí jedna lze zvolit řešení A i B. Poslední zmíněný krok výpočtu předpokládá, že předchozími kroky došlo k odfiltrování všech špatných řešení a zbyly jen takové, které si jsou rovny. V aplikaci tento výběr provedeme pomocí náhodného výběru.

4.4 Nástin použitého řešení

Ve výše uvedené kapitole byl přiblížen způsob, jakým algoritmus dojde k vyhledání a následnému výběru jednoho konkrétního tahu. Tato kapitola se již věnuje samotné realizaci tohoto

algoritmu. Základem výpočtu bude *herniPole*, která obsahuje informace o stavu jednotlivých políček. Z důvodu urychlení výpočtu však bude vhodné souřadnice figurek udržovat v pomocné struktuře. Podle určených pravidel obsahuje hra deset figurek pro každého hráče. Souřadnice figurek budou již při začátku vloženy do tohoto seznamu. Třída, která je bude uchovávat, je pojmenovaná příhodně *SeznamFigurek*. Z návrhu bylo patrné, že každá figurka bude obsahovat několik různých řešení, pro pořádek tuto třídu vystavěnou nad lineárním seznamem pojmenujeme *SeznamReseni*. Jedním řešením je myšlena posloupnost souřadnic, přes které musí figurka svůj tah projít. Toto řešení by mohlo obsahovat pouze počáteční a cílovou souřadnici. Zobrazovací vrstva by však poté přesunula figurku rovnou a uživatel by ztratil přehled, jakou cestou se na výsledné pole přemístila. Tuto posloupnost budeme také uchovávat v lineárním seznamu pojmenovaném *SeznamTahu*. Pro souřadnice již není potřeba vytvářet další třídu, návrh ji již obsahuje. [15]

Algoritmus výpočtu bude zastřešovat třída *TahPocitace*. Vzhledem k její obsáhlosti bude delegovat výpočet jednotlivých řešení třídě *VypocetTahuFigurky*. Start výpočtu je spuštěn zavoláním metody *spocitejTah()*. Této třídě vždy předá jednu instanci figurky. Na základě souřadnic třída pomocí rekurzivní funkce projde všechny možnosti, kam by se figurka v daném tahu mohla pohybovat pomocí skoku. Pokud je možné některý ze skoků provést, metoda tah uloží jako možné řešení a rekurzivně pokračuje dál pro zjištění, jestli neexistuje možnost skok zřetězit dalšími. Všechna řešení, která najde, uloží do seznamu řešení, které obsahuje samotná figurka. Poté, co jsou takto nalezeny všechny možnosti, kam by se figurka mohla pomocí skoků přemístit, nastává druhá fáze výpočtu, při které třída prochází možnosti, kam se figurka může přesunout pomocí posunu o jedno pole. Třída neuvažuje nad možností tahu směrem vzad, jelikož ten by ji cíli nepřiblížil. U pohybu skokem však toto pravidlo neplatí, při zřetězení skoků je možné provést tah vzad, a poté dalšími skoky zase vpřed. Po naplnění seznamu řešení podpůrná třída svůj výpočet končí. Takto jsou pomocí cyklu foreach nalezena všechna řešení pro všechny počítačové figurky. Tato řešení jsou uložena v instancích figurek v lineárním seznamu pojmenovaném *SeznamReseni*.

Po ukončení výpočtu nastává druhá fáze, a to ohodnocení těchto řešení. Tento postup je spuštěn zavoláním metody *provedTah()*. Třída již sama projde všechna řešení všech figurek zadaných ve svém seznamuFigurek. Každé řešení je nezávisle ohodnoceno metodou *ohodnot_reseni*. Postup ohodnocení je poměrně obsáhlý, proto se mu bude věnovat samostatná kapitola.

Následuje třetí fáze, výběr nejlepšího tahu. Při ohodnocování figurek si metoda zapamatovala nejlepší hodnocení, které některý z tahů obdržel. Podle tohoto ohodnocení následně provede výběr všech řešení, která danému hodnocení odpovídají. Při zadání tolerance navíc umožňuje i pomyslnou laťku snížit a výběr tím zvětšit. Všechna tato řešení jsou uložena v proměnné *seznamMoznychTahu*.

Po ukončení výběru následuje poslední fáze, a to výběr náhodného tahu z podmnožiny řešení uložených v proměnné *seznamMoznychTahu*. Vybrané řešení je reprezentováno instancí třídy *SeznamTahu*. Tento seznam obsahuje množinu souřadnic, tzv. cestu, přes kterou daný tah pro-

vede. Metoda tedy o této cestě informuje samotný model (*HerniPole*). Tomuto modelu také řekne o změně souřadnice počítačové figurky. Tyto souřadnice je také nutné upravit v proměnné *seznamFigurek* ve třídě *TahPocitace*.

5 Vzhled aplikace

5.1 Rozvržení herního okna

Aby bylo možno začít s tvorbou pravidel pro lidského hráče, bude nutno nejprve vytvořit grafické zobrazení aplikace. Jinak řečeno, bude nutné vytvořit uživatelské rozhraní, kterému se zkráceně říká GUI (Graphical User Interface). Základem grafické aplikace je tzv. *MainFrame*, objekt, který je odvozen od třídy *JFrame*. Jak již z názvu třídy vyplývá, jedná se o rámeček, který do sebe zapouzdřuje ostatní objekty a udržuje je pohromadě. Lze ho tedy nazvat hlavním grafickým oknem aplikace. [14]

Vzhled aplikace je vyobrazen na obrázku č. 6. Toto základní GUI obsahuje herní plochu a tlačítko na ukončení tahu hráče. Dá se předpokládat ještě přidání menu, ve kterém bude mít hráč možnost spustit novou hru, či ukončit aplikaci.

Z pravidla není dobré kreslit přímo na plochu hlavního rámečku. A to např. z důvodu, když by bylo zapotřebí aplikaci nějakým způsobem rozšířit, např. o panel s časomírou. Bylo by nutné umístění objektů na ploše upravit a v tomto případě by to i znamenalo poupravit celý kód samotného vykreslování. Herní plocha tam bude vykreslena na svém samotném panelu, objektu odvozeném od třídy *JPanel*. Ke kreslení na plochu tohoto panelu využijeme metodu *paintComponent(Graphics g)*. Na rozdíl od jiných metod, tato je zavolána při každém překreslení panelu aplikace. [13]

V této aplikaci není předpokládána možnost zvětšování ani zmenšování herního pole. Z tohoto důvodu se pro velikost okna mohou použít fixní hodnoty. V aplikaci byly použity rozměry:

- Šířka herního pole = 650px.
- Výška herního pole = 650px.

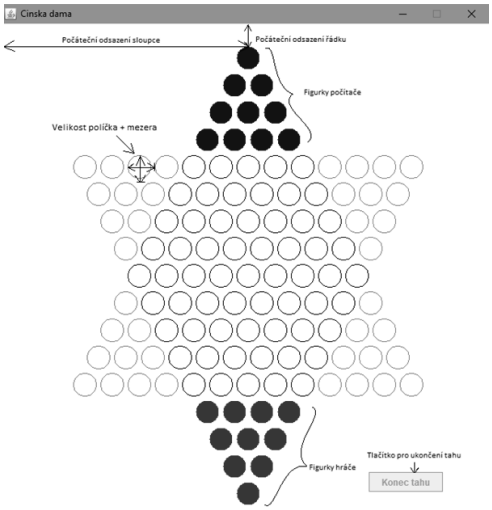
Velikost panelu tedy bude odpovídat velikosti herního pole. K výšce se bude ještě přičítat 30px, z důvodu panelu s názvem. Rozměry hlavního okna budou následující:

- Šířka herního pole = 650px.
- Výška herního pole = 680px.

Po grafickém rozvržení i určeném způsobu vykreslení panelů a tlačítek, se budou tyto prvky postupně vkládat do hlavního okna aplikace. Jako poslední příkaz se v konstruktoru bude nacházet *setVisible(true)*, tímto příkazem se zobrazí vytvořená aplikace a bude zpřístupněna pro koncového uživatele.

5.2 Vykreslení herní plochy

V této podkapitole budou podrobněji popsány způsoby vykreslování vzhledu aplikace. Ze všeho nejdříve se nadefinují velikosti jednotlivých částí. Šířku a délku celé plochy (jak již bylo řečeno,



Obrázek 6: Herní plocha

v aplikaci je použita velikost 650px), velikost políčka (v aplikaci je použita velikost 30px). Pro lepší přehlednost bude mezi políčka vkládána také mezera (ta má hodnotu 6px). Důležitým údajem je také počáteční bod, od kterého bude začínat vykreslování jednotlivých pixelů.

Vykreslování bude probíhat od shora směrem dolů. Nejvýše je umístěn vrchol hvězdy, který se navíc nachází uprostřed prvního řádku. Proto bude počáteční bod souřadnice sloupce (souřadnice na ose x) určen jako poloviční šířka plochy s odečtením poloviční šířky políčka (středová osa řádku). Souřadnice řádku (souřadnice osy y) bude také o několik pixelů posunuta, aby vykreslení neprobíhalo hned od okraje herního okna (jedná se pouze o zpřehlednění celkového výstupu grafického zobrazení). Také jednotlivé mezery mezi políčky hráči umožní snadnější orientaci na herní ploše.

Na každém řádku je jiný počet políček. Z toho důvodu bude vytvořeno pole, obsahující informace ohledně počtu políček z jednotlivých řádků.

$$počátečníSouřadniceSloupce = (šířkaPlochy/2) - (šířkaPolíčka/2)$$

Vykreslení se bude provádět pomocí dvou vnořených cyklů s řídící proměnnou (for), stejně jako při práci s klasickým dvourozměrným polem. Využije se zde připravené pole s počty políček z každého řádku. První cyklus, který je pro řádky, bude mít počet opakování podle velikosti tohoto pole. V druhém cyklu, řešícím sloupce, se počet opakování zvolí podle hodnoty v daném poli, která udává počet políček na aktuálním řádku. Číslo řádku, na kterém se v cyklu zrovna nachází, je určeno proměnnou z prvního cyklu. Toto číslo také určuje index, na kterém se nachází informaci o počtu políček na daném řádku.

Pro sudé a liché řádky (počítáno od 0), bude souřadnice sloupce pro vykreslení vypočítávána jinak, jelikož jsou liché řádky o nějakou část posunuty oproti sudým.

Souřadnice řádku se počítá stejně v lichých i sudých řádcích, a to tak, že k zvolené počáteční

souřadnici řádku pro vykreslení bude přičítat výška políčka i s mezerou, která bude vynásobena aktuální hodnotou proměnné z prvního cyklu. Budeme počítat s odsazením od okraje 30px (počáteční hodnota vykreslování řádku). Ve vzorci bude proměnná z cyklu označena jako n. Vzorec bude vypadat následovně.

$$SouřadniceŘádku = 30 + (výškaPolíčka + mezera) * n$$

Souřadnice sloupce v sudých řádcích se bude vypočítávat tak, že k počáteční souřadnici pro vykreslování sloupce se bude přičítat šířka políčka i s mezerou, vynásobená proměnnou z druhého cyklu. Následně se od této hodnoty odečítá poloviční délka aktuálního řádku vynásobená šířku políčka i s mezerou. Pro liché řádky se bude připočítávat k tomuto ještě poloviční šířka políčka i s posunem. Ve vzorci nám m značí proměnnou z druhého cyklu, $š$ je šířka políčka (i s mezerou), p znázorňuje počáteční souřadnici pro vykreslování sloupce, délka řádku bude označena jako d.

$$SouřadniceSloupceSudéhoŘádku = (p + š * m) - (d/2) * š$$

$$SouřadniceSloupceLichéhoŘádku = (p + š * m) - (d/2) * š + š/2$$

Bude potřeba také rozlišovat políčka, na kterých jsou umístěny figurky, volná políčka a případně také ta, která tvoří ostatní cípy hvězdy (ty které nejsou v počátečním stavu vyplněny figurkami hráče ani počítače). K tomuto je vhodné použití switche, který se bude rozhodovat podle hodnoty v herním poli. Jelikož souřadnice sloupce políčka v tomto poli je jiná, než při samotném vykreslování, bude se muset tato souřadnice pro herní pole vypočítat. Výpočet bude vypadat tak, že se od počtu políček z nejdelšího řádku (což je třináct) odečte počet políček z aktuálního řádku, k tomuto bude přičtena hodnota proměnné z druhého cyklu vynásobenou dvěma. Tato proměnná je označena jako m.

$$IndexSloupceVHernímPoli = 13 - početPolíčekNaŘádku + m * 2$$

Počet políček z nejdelšího řádku udává šířku pole. Odečtením počtu políček z daného řádku se zjistí posun oproti klasickému vykreslování od okraje pole. K výsledku se přičte aktuální index sloupce, který je vynásoben dvěma. Násobení dvěma je zde z důvodu, že pole, se kterým se pracuje, obsahuje neplatná políčka, tudíž je jeho velikost dvakrát větší. Tímto vzorcem je tedy vypočítán index sloupce. Jelikož index řádku je již známý z předešlého výpočtu, algoritmus tak již zná kompletní souřadnici indexu herní plochy.

Každá hodnota políčka bude vykreslována jinou barvou. Pro prázdná políčka bude zvolena černá barva, modrá pak pro počítač, červená pro hráče. Ostatní cípy hvězdy, které nejsou na začátku hry obsazeny figurkami, budou například vykresleny šedou barvou. Pro prázdná políčka a políčka značící ostatní cípy hvězdy bude pro přehlednost použito pouze vykreslení okraje. Toto

vykreslení je realizováno metodou *drawOval*. Pro figurky pak bude zvoleno vybarvení celé plochy políčka, pomocí metody *fillOval*.

6 Vytvoření logiky pro lidského hráče

6.1 Komunikace s oknem aplikace

Prozatímní grafické rozhraní tedy již umí vykreslit herní plochu, jednotlivá políčka i figurky. Tato kapitola se zaměří na vytvoření funkcionalit, které zajistí komunikaci mezi uživatelem a aplikací. Tato komunikace bude prováděna pomocí jednotlivých kliků myši v okně aplikace. Hra půjde ovládat jen za pomoci myši, obsluhovat klávesnicí nebude potřeba. Proto, aby aplikace zachytávala akce myši, je potřeba připravit vhodný posluchač událostí (tzv. event listener). Na zachytávání jednotlivých kliků myši se však dá využít služeb hned několik různých posluchačů, nejpoužívanějšími jsou zejména *mouseClicked* nebo *mousePressed*. Hlavním rozdílem těchto dvou posluchačů je, že zatímco *mousePressed* se zavolá ihned při stisknutí tlačítka myši, *mouseClicked* až poté, co uživatel tlačítko pustí. [14] [19]

V tomto případě bude výhodnější využití *mousePressed*. Zamezí se tímto problému s rozpoznáním místa, kdy uživatel zároveň s kliknutím ještě posune kurzor myši na sousední políčko. V tomto případě by u metody *mouseClicked* došlo k označení druhého políčka, což je pro uživatele matoucí. [19]

Hlavním úkolem tohoto posluchače bude zachytávat kliky myši a rozeznat, na které políčko herní plochy chtěl hráč kliknout. Již bylo řečeno, že kliknutí mimo herní plochu se bude moci ignorovat. Hrací plocha není rozprostřena přes celou šířku okna aplikace, ale je umístěna v jejím středu. Vzdálenost mezi okrajem aplikace (panelu) a začátkem herní plochy se bude nazývat posun (názorně zobrazeno na obrázku č. 6, kde je tento posun označen jako počáteční odsazení sloupce/řádku). Pokud tedy metoda rozpozná kliknutí mimo herní plochu (vnitřní obdélník), akci bude ignorovat.

Pokud došlo ke kliknutí dovnitř hrací plochy, musí se tyto souřadnice kliknutí x a y v pixelech převést na souřadnice herní plochy. Tento přepočet je poměrně jednoduchý.

Index řádku se vypočte tak, že se od souřadnice řádku (bodu, na který se kliklo), odečte výše zmíněný posun od okraje a poté se vydělí velikostí políčka i s mezerou (budeme ve vzorci rovnou počítat, že k velikosti políčka je připočtena i mezera).

$$IndexRadku = (SouradniceRadku - posunRadku)/vyskaPolicka$$

Výpočet indexu sloupce je komplikovanější. Herní plocha ve tvaru hvězdy totiž má všechny řádky konstantně vysoké. U sloupců však toto pravidlo neplatí. U indexu sloupce se bude muset rozlišit, zda se jedná o sudé, či liché řádky. Počáteční souřadnice sloupce se zde bude značit p, šířku políčka i s mezezou se označí jako $š$.

Pro sudé řádky platí:

$$IndexSudehoSloupce = ((SouradniceSloupce - (p - (6 * š)))/š) * 2$$

Sudý řádek bude mít index vypočten tak, že se od bodu kliku, značícího sloupec, bude odečítat rozdíl posunu vykreslení sloupce. Následně se bude velikost políček i s mezerou násobit poloviční velikostí maximálního počtu políček na řádku (zaokrouhleno směrem dolů). Výsledek poté bude vydělen velikostí políčka i s mezerou. Celé se to pak vynásobí dvěma. Vynásobení dvěma je zde opět z důvodu, že řádek v našem poli je dvakrát delší než ve vykreslování.

Pro liché řádky naopak platí:

$$IndexLichehoSloupce = ((SouradniceSloupce - (p - 6 * š) - š/2)/š) * 2 + 1$$

U lichých řádků bude muset být od souřadnice bodu kliku navíc odečtena poloviční velikost políčka i s mezerou. K celému výsledku se navíc přičte jednička, jelikož jsou tyto řádky o jedno políčko posunuty.

6.2 Kontrola tahu

Poté, co HerniPanel ověří, že klik proběhl dovnitř herní plochy, pošle skrz MainFrame controlleru informaci o provedení této akce. Součástí této zprávy jsou souřadnice, na které uživatel kliknul. Zobrazovací vrstva tak pouze informuje, že ke kliknutí došlo. Jaká činnost bude kliknutím vyvolána, je v režii řídícího controlleru.

Fáze hráčova tahu se dají popsat následujícími kroky:

1. Hráč provede výběr figurky pro provedení svého tahu.
2. Hráč provede označení políčka, na které se má figurka přesunout.
3. Controller provede přesun figurky z původního pole na cílové.
   - 3a) Pokud v bodu 3 došlo k posunu figurky na sousední pole, pokračuj na bod 4.
   - 3b) Pokud v bodu 3 došlo k provedení skoku, vrať se do bodu 2. Umožni hráči přechod na bod 4 pomocí tlačítka "Konec tahu".
4. Konec hráčova tahu.

Hráč má možnost v druhé fázi svého tahu provést kliknutí opět na stejnou figurku jako ve fázi jedna. Tímto krokem dojde k odznačení figurky a algoritmus tak pokračuje od začátku. Pokud dojde ve fázi 3 k provedení skoku, musí hrát ve fázi 2 vybrat pouze pole, na které se může figurka opět přenést pomocí skoku. V jednom tahu nelze kombinovat skok s posunem. Pokud v jakoukoliv fázi dojde o označení políčka, na které tah nelze provést, akce je ignorována.

V programu je toto řízení fází řešeno metodou *kliknutiMysi*, kterou obsahuje řídící controller. Pro uchování označené figurky z první fáze využívá proměnnou *souradniceKliknuti*, která patří samotnému modelu, *Hernímu Poli*. Tuto proměnnou též využívá zobrazovací vrstva aplikace. Pro přehlednost je označená figurka obtažena černým rámečkem.

6.3 Tahová logika

Čínská dáma patří mezi tahové hry. Na tazích se vždy jednotliví hráči střídají obvykle v pořadí určeném podle směru hodinových ručiček. V případě naší aplikace, která umožňuje hru pouze dvou hráčů, bude hru vždy začínat lidský hráč. Průběh hráčova tahu byl popsán v předchozí podkapitole. Důležité je zmínit, že zatímco po posunu figurky na sousední pole dochází k automatickému ukončení tahu, při provedení skoku tomu tak není. Po provedení skoku musí hráč tah ukončit ručně. Při hraní na deskové hře obvykle končí tah hráče ve chvíli, kdy hráč svoji figurku upustí. Toto chování, kdy by uživatel musel v průběhu svého tahu držet stisknuté tlačítko myši, by bylo uživatelsky nepřívětivé. Hra obvykle není zatížena časovým limitem, který by měl hráč na provedení tahu. Naše aplikace ho také nebude obsahovat.

Poté, co se hráč rozhodne jedním ze způsobů ukončit svůj tah, tak řídící controller nejprve zkontroluje, jestli provedeným tahem nedošlo k vítězství hráče. To lze jednoduše zjistit ověřením, zdali jsou všechny jeho figurky na cílovém poli. Pokud ke konci hry nedošlo, controller předá řízení třídě *TahPocitace*, která pomocí metody *spocitejTah()* vypočítá seznam všech možných tahů počítače (viz kapitola č. 4.3). Následným zavoláním metody *provedTah()* se provede ohodnocení množiny nalezených řešení, následnému výběru nejlepších tahů, a také provedení jednoho konkrétního. Poté, co počítač tímto svůj tah ukončí, dojde k opětovné kontrole herního pole. Tentokrát se však ověřuje, jestli nedošlo k posunu všech počítačových figurek do jeho cílového pole. Pokud ne, následuje opět hráčův další tah. Takto se hra opakuje až do doby, než jeden z hráčů přesune všechny své figurky do cíle. Oznámení konce hry bude provedeno například zobrazením dialogového okna.

Existuje však možnost, že ve hře dojde k remíze. K té může dojít v případě, že jeden z hráčů zablokuje pohyb soupeřově figurce v jeho původním domečku. V naší aplikaci řešíme remízový stav v případě, kdy pro počítač neexistuje žádný tah, který by ho přiblížil k cíli, případně zůstal vůči cíli neutrální.

Průběh tahů tedy lze shrnout do těchto bodů:

1. Tah hráče.
2. Ověření, zdali se hráč nestal vítězem partie.
3. Tah počítače.

   3a V případě, že počítač nemá žádný možný tah, došlo k remíze.
4. Ověření, zdali se počítač nestal vítězem partie.

7 Rozbor první verze odehrané hry

Po předchozích kapitolách je již aplikace ve funkčním stavu. Umožňuje jednak reagovat na podněty hráče, tak i odehrávat tahy za pomocí jednoduché umělé inteligence. V této kapitole tedy bude popsáno, jak vypadá ukázková hra s touto verzí aplikace. V popisu bude kladen důraz především na tahy umělé inteligence, která v tuto fázi vybírá nejlepší tah jen na základě toho, která figurka ho posune nejblíže k cíli. Zjednodušeně lze hodnotící funkci napsat jako porovnání řádků souřadnic počátečního a koncového tahu:

$$hodnocení = koncovýřádek - počátečnířádek$$

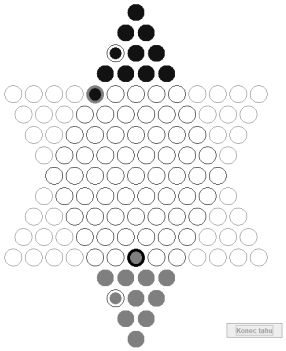
Předpokládaný výběr tahů tedy bude následující:

1. zřetězené skoky,
2. jednoduché skoky,
3. posuny směrem dolů,
4. posuny směrem do stran.

Tahy se záporným ohodnocením, které vzniknou tažením figurky vzad, by měly být ignorovány. Výběr tahů také neobsahuje žádnou toleranci. Algoritmus výpočtu také neobsahuje predikování tahů dopředu. Funguje tedy jen na možnostech aktuálního tahu, proto nelze očekávat, že by si například postupně připravoval dlouhý skok figurkou přes polovinu herní plochy.

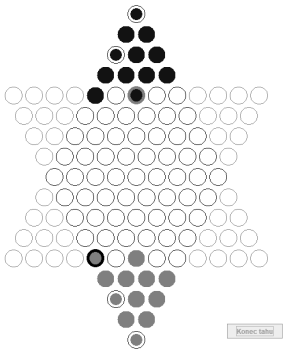
Na následujících řádcích a obrázcích budou ukázány jen vybrané zajímavé tahy.

Podle těchto pravidel, by měl počítač ve svém prvním tahu zahrát s jednou z figurek, která se nachází na třetím řádku. Jako jediné totiž mají možnost skoku. Na obrázku č. 7 je zobrazeno, že algoritmus takový tah opravdu zvolil. Na následujících obrázcích menší vyplněné kolečka značí políčka, ze kterých, či přes které, byl proveden tah. Konečná pozice figurky je zobrazena zvýrazněným okrajem.



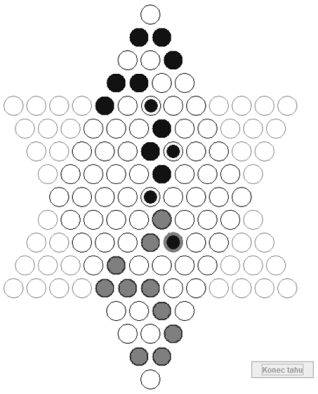
Obrázek 7: Začátek hry

Následuje odehrání tahu hráče, který ještě nemohl výpočet tahu počítače ovlivnit. Při druhém tahu lze očekávat pouze jeden možný tah počítače, a to provedení zřetězeného skoku figurkou hned z první řady. Tato situace ve hře také nastala a je zobrazena na obrázku č. 8. Z těchto tahů lze usuzovat, že algoritmus správně ohodnocuje delší skoky a tím je i upřednostňuje před kratšími. K posunu figurkou o jedno pole také správně nedošlo.



Obrázek 8: Řetězení skoků

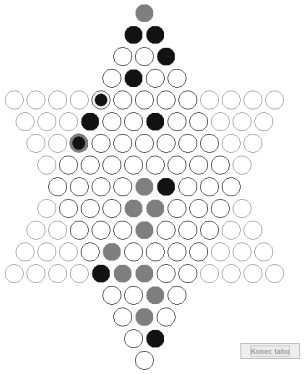
Po provedení několika tahů se situace dostala do stavu, který je zobrazen na obrázku č. 9 Na něm algoritmus vybral tah, který přeskakuje nejen své figurky, ale korektně i soupeřovy. Tahy jako takové prozatím nejsou voleny špatně, ale již začíná být patrné, že při dalších tazích pravděpodobně dojde k postupnému osamocení zadních figurek.



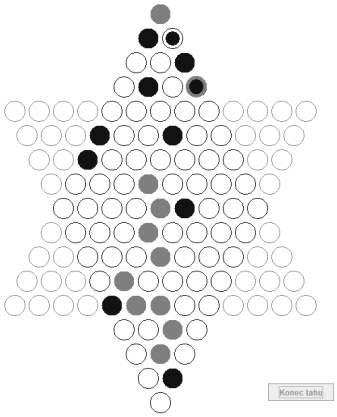
Obrázek 9: Řetězení skoků i přes soupeřovy figurky

Situace se po několika dalších tazích dostala do stavu na obrázku č. 10. V tomto tahu jsou figurky počítače od sebe poměrně vzdálené. Došlo i k postupnému osamostatnění některých z nich. Pro výběr tahu dle zadaných pravidel existují dvě řešení. Skok s figurkou z druhého řádku, či skok s figurkou v levé části herní plochy. Jelikož je hodnotící funkce považuje za stejně dobré, algoritmus z nich vybral jeden náhodný, v tomto případě druhý zmíněný. Situace se

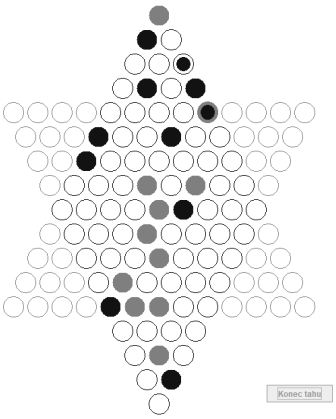
opakuje i v dalších následných tazích, které jsou vyobrazeny na obrázcích číslo 11 a 12. Z nich je také patrné, že algoritmus z nich opravdu vybíral náhodně.



Obrázek 10: Ukázkový tah hry

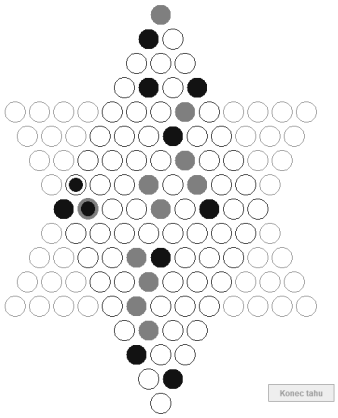


Obrázek 11: Ukázkový tah hry



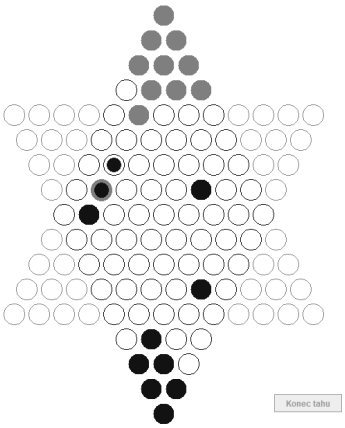
Obrázek 12: Ukázkový tah hry

Zajímavá situace je také zobrazena na obrázku číslo 13. Zde již došlo ke stavu, kdy pro počítače neexistuje žádný tah, kterým by mohl vykonat skok a ten ho přiblížil k cíli. Nejlepší tah tak má ohodnocení rovné jedné (posun o jeden řádek směrem k cíli). Algoritmus k tahu náhodně vybral figurku opět v odlehlé levé části. Ze situace je patrné, že přiblížení se k okrajům plochy podél cípů nehrajících hráčů vede k osamocení a postupné ztrátě tempa. Z obrázku je také patrné, že tah do cípu nehrajících hráčů je zakázaný a algoritmus ho tedy správně nepoužil. V opačném případě by zvolil skoku do tohoto cípu.



Obrázek 13: Skok na okraji herní plochy

Ztráta tempa způsobena skoky v levé části herního pole se ukázala jako klíčová a vedla k vítězství lidského hráče. Konečný stav hry těsně před odehráním hráčova tahu je zobrazen na obrázku 14 Dále bychom mohli předpokládat, že by následující tahy počítače vedly pomocí skoků v levé části. Poslední dvě figurky ze středu plochy by pravděpodobně do cíle byly přesunuty pomocí posunů o jedno políčko.



Obrázek 14: Konec hry

8 Optimalizace výběru tahů

Z výše uvedené partie lze zpozorovat, že provedení skoku a přiblížení se cíli pouze na základě souřadnic řádku není vždy nejefektivnějším tahem. Postupnými skoky vedenými po kraji plochy se sice počítač přibližuje k cíli vždy o dva řádky naráz, své figurky ale tímto osamostatní. Tím se počítač dostane do situace, kdy nebude moci používat zřetězení skoků. V pozdějších fázích hry tak bude mít skupiny osamocených figurek rozdělené po dvojicích, které se přibližují cíli jen skoky mezi sebou. V krajních polích se navíc nedá očekávat, že by do nich umístil figurky i jeho protivník. Z těchto údajů vyplývá, že by algoritmus měl upřednostňovat tahy, které jeho figurky vedou ku středu hracího pole.

Prvotní optimalizace tahů spočívala v úpravě výběru tahů. U každé figurky se výběr upravil tak, aby obsahoval jen ta řešení, která dovedou figurku blíže ke středu. Toto profiltrování bylo provedeno až těsně před náhodným výběrem tahů. Chování počítače se vylepšilo, ale k výše uvedeným stavům stále docházelo. Pokud algoritmus zjistil, že skok lze provést pouze jednou jeho figurkou, provedl ho. Tím se opět dostal do situace, kdy v postupných tazích dvojicí figurek došel do levého, či pravého kraje plochy. Pokud také existoval dlouhý skok přes hráčovy figurky, algoritmus ho vyhodnotil jako správný. Ohodnocení totiž nepočítalo s tím, že se tímto skokem jeho figurka dostane do osamoceného místa v kraji herního pole. V takovém případě by mohlo být výhodnější tah přerušit jen po provedení jednoho skoku, nikoliv po celém zřetězení.

Z hlediska výkonu aplikace se také ukázalo, že není potřeba nejprve vyhledávat samotné skoky a až poté výběr rozšiřovat o posuny. Hodnotící funkce pro oba tyto výběry byla totiž stejná. Pravidla tak půjde výrazně zjednodušit.

Nynější pravidla se tak budou držet původního návrhu, kdy dojde nejprve k nalezení celé množiny počítačových tahů. Následuje ohodnocení a výběr jednoho konkrétního tahu z této množiny řešení. K tomu dojde k úpravě hodnotící funkce, která bude mít za úkol zamezit právě postupným tahům k okrajům plochy. Pokud tah směruje na volné pole, které je od středu vzdálené více jak tři políčka, tah dostane penalizaci. Tato penalizace se zvyšuje s postupným oddalováním se od středu. Po této úpravě hodnocení již také není nutné mít tahy předem roztříděné podle takových, které figurku vedou směrem ke středu pole. Jakýkoliv tah, který vede na jedno z polí u středu, totiž penalizován není. Hodnocení je tedy upraveno do vzorců:

$$vzdalenostOdStredu = \|(cilovySloupec - stredovySloupec)\|$$

$$penalizace = \begin{cases} vzdalenostOdStredu, & vzdalenostOdStredu > 3. \\ 0, & \text{jinak.} \end{cases}$$

$$hodnoceni = cilovyRadek - pocatecniRadek - penalizace$$

Nynější kompletní výběr tahů by se dal shrnout takto:

1. Vyhledají se všechny možné tahy.

2. Ohodnocení všech tahů.

3. Výběr tahu z nejlépe ohodnocených

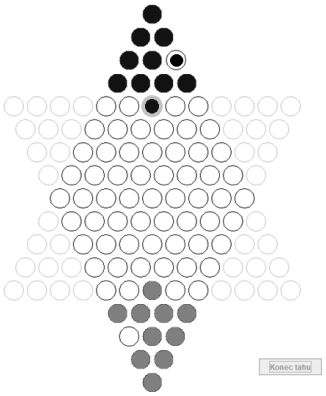
4. Provedení vybraného tahu

Do této metody se dají vložit různé podmínky, které mohou k výslednému hodnocení přidat určitý bonus, či naopak udělit penalizaci. Příkladem může být bonus pro figurky, které jsou příliš pozadu na herní ploše a hrozí jim tak osamocení. Naopak penalizace může být udělena figurkám, které jsou již umístěny v cílovém poli.

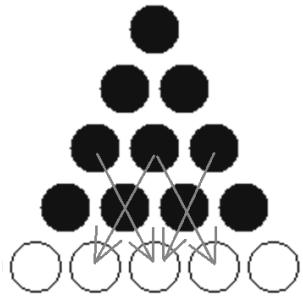
Jak tato úprava pomohla počítači, si přiblížíme v následující kapitole, která bude věnovaná analýze odehrané partie s upravenou hodnotící funkcí.

9 Rozbor druhé verze odehrané hry

V této kapitole budou ukázány tahy ze hry, ve které již platí pravidla popsaná v optimalizaci tahů. Na obrázku č. 15 je vyobrazen klasický začátek hry. Počítač pro svůj tah zvolil správně figurku z třetího řádku. S jinými figurkami by skok provést nemohl. Jelikož platí také penalizace za skok do kraje, nemělo by se stát, že by v prvním skoku vybral skok vpravo dolů u figurky, kterou tentokrát táhnul. Na obrázku č. 16 jsou vyobrazeny všechny tahy, které může počítač pro svůj první tah zvolit.



Obrázek 15: Počáteční tah

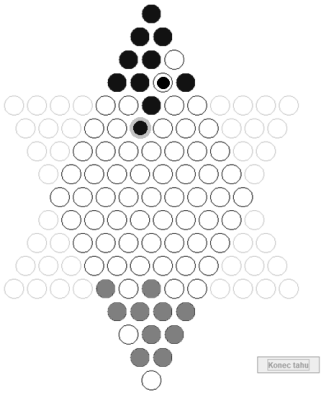


Obrázek 16: Možnosti na začátku hry

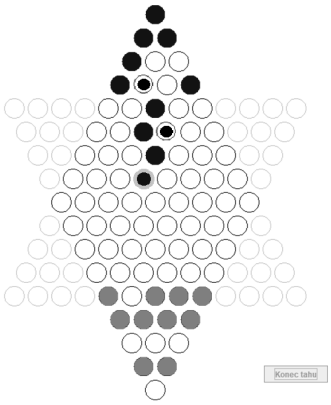
Již v druhém tahu lze pozorovat, že počítač lépe vyhodnotil obyčejný skok přes figurku, než zřetězení dvou skoků. Zřetězeným skokem po pravém okraji hvězdy, by se počítač posunul blíže k domečku. Zároveň by se však posunul dál od středové části herní plochy, což je stav, kterému se optimalizací výběru tahů snažilo zabránit. Dokonce zde nastala situace, kdy počítač zvolil skok s figurkou z prvního řádku. Tento skok však ukončil již po přeskočení první figurky i ve chvíli, kdy měl možnost pokračovat ještě jedním skokem. Tato situace je vyobrazena na obrázku č. 20.

Na obrázcích č. 18 a 19 lze shlédnout příklady zřetězených skoků v této partii.

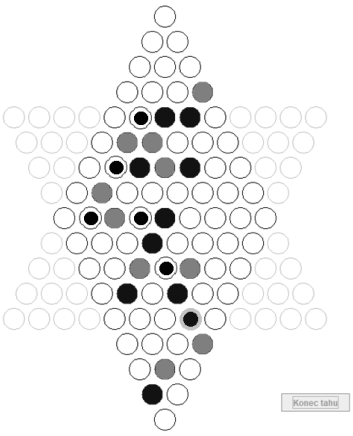
Figurky počítače se nepohybovaly příliš daleko od prostředního sloupce herní plochy, což velmi pomohlo celkovému výběru tahů. Tuto partii díky tomu počítač vyhrál (viz obrázek č. 21). Počet tahů byl v této partii téměř totožný jako u naší předchozí ukázkové hry. Přesto zde již



Obrázek 17: Skok

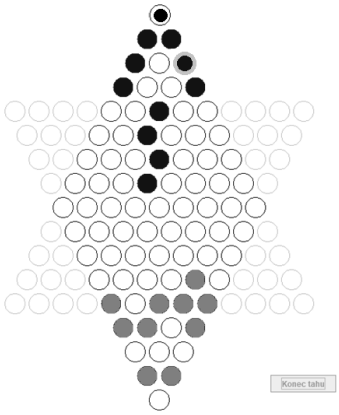


Obrázek 18: Zřetězení skoků

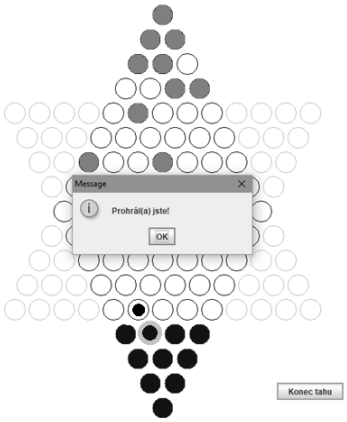


Obrázek 19: Zřetězení skoků

počítač hru úspěšně dokončil, zatímco u minulé hry měl ještě několik figurek poměrně daleko od domečku.



Obrázek 20: Nedokončené zřetězení skoků



Obrázek 21: Konec hry

9.1 Simulace her proti jiné umělé inteligenci

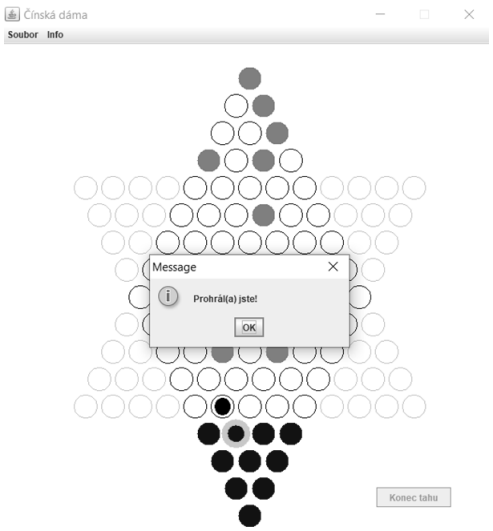
Pro otestování programu bylo vyzkoušeno i odehrání několika partií ve chvíli, kdy proti sobě hráli jen dva počítačoví hráči. Dvě různé aplikace nelze proti sobě postavit přímo. Pro toto testování tedy bylo nutné mít rozehrané dvě partie Čínské dámy. První partie byla rozehrána v aplikaci Chinese Checkers od autora Zheng Yi, která je k dispozici volně ke stažení z obchodu Google Play. [10] Druhá partie byla rozehrána v aplikaci vytvořené v této práci. U volně dostupné hry byla využita možnost, kdy počáteční tah provedl počítač, nikoliv hráč. Po provedení prvního tahu byl stejný tah ručně zadán do námi vytvořené aplikace. Na tento tah zareagovala námi zkonstruovaná umělá inteligence. Tento tah byl ručně přenesen zpátky do původní aplikace. Žádný tah tedy nebyl ovlivněn lidským faktorem. Celou dobu zápasu tak existovaly dvě stejné partie, ve kterých se rozhodovalo o vítězi. Lidský faktor pouze přenášel tahy mezi aplikacemi.

Námi vytvořená umělá inteligence byla schopna porazit protější umělou inteligenci od autora Zheng Yi hned v několika zápasech. Z pěti odehraných her se stala vítězem čtyřikrát. Po provedení tohoto testu se dá říci, že námi vytvořená logika tahů počítače je dostačující pro základní

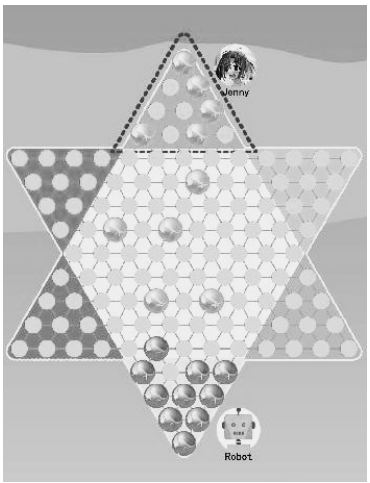
funkčnost hry.

Dostupné materiály aplikace Chinese Checkers od autora Zheng Yi neuvádějí, jakým způsobem své tahy volí. Subjektivně lze říci, že aplikace využívá jistou formu predikce, ale přesný postup výběru tahů není znám.

Zakončení jedné z her je ukázáno na obrázcích č. 22 a 23. [10]



Obrázek 22: Konec simulované hry AI proti AI z pohledu naší aplikace



Obrázek 23: Konec simulované hry AI proti AI z pohledu android aplikace [10]

10 Realizace projektu

10.1 Obecný popis struktury programu

V této části budou popsány třídy vytvořeného programu. Pro náhled na jednotlivé třídy a jejich vzájemné vazby se nabízí zobrazení za pomoci diagramu tříd. Vzhledem k obsáhlosti diagramu jsou na následujícím obrázku č. 24 zobrazeny pouze nejdůležitější vazby. Jednotlivé třídy budou popsány zvlášť níže.[20]

10.2 Modelová vrstva aplikace

Do datové struktury programu, jinak řečeno do modelové části, patří tyto třídy:

- *Figurka*,
- *HerniPole*,
- *Souradnice*,
- *SeznamFigurek*,
- *SeznamReseni*,
- *SeznamTahu*.

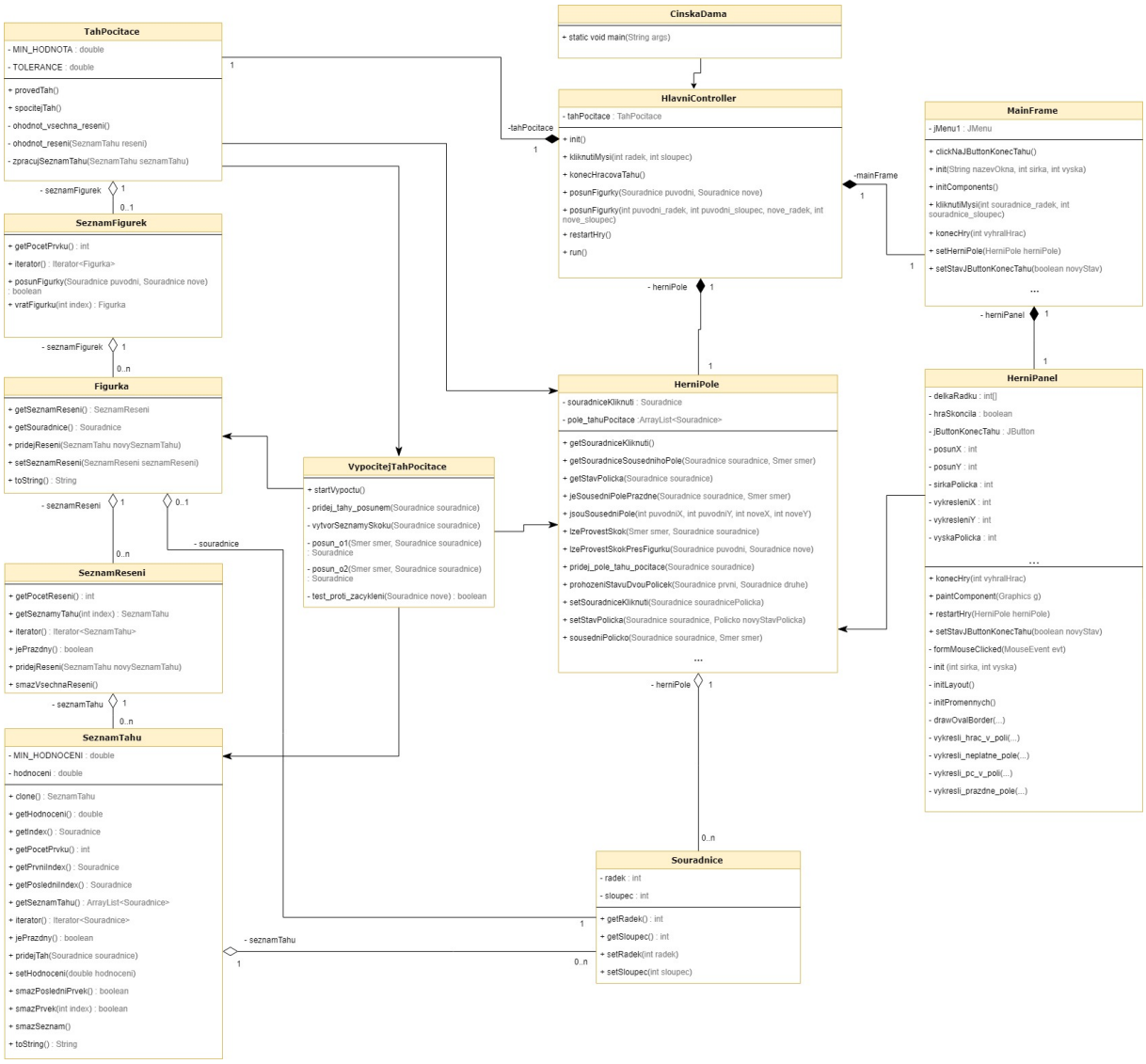
Třídy ke své funkčnosti využívají ještě tři výčtové typy:

- *Smer*,
- *Tah*,
- *Policko*.

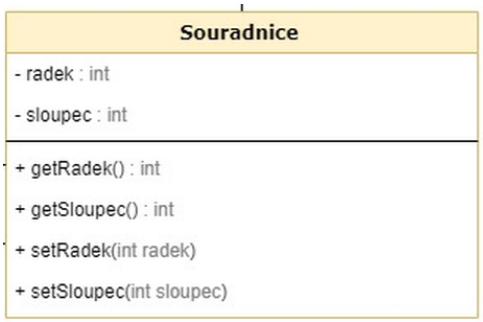
Nejjednodušší entitou modelu je třída *Souradnice* (viz obrázek č. 25). Jejím úkolem je uchovávání indexu řádků a sloupců. Většina ostatních modelů ji tak využívá k adresaci jednotlivých políček na herním poli. Třída obsahuje vlastní implementaci zděděné metody *Equals*, která říká, zdali jsou dva různé objekty totožné. V tomto případě považujeme za stejné ty souřadnice, které odkazují na stejný řádek i sloupec. [18]

Třída *HerniPole* je základním stavebním prvkem celé aplikace (viz obrázek č. 26). Obsahuje informace o stavu a umístění všech políček celé herní plochy. Pro jejich adresaci využívá výše zmíněnou třídu *Souradnice*. Každé takovéto políčko se může nacházet hned v několika stavech. Ty jsou definované pomocí stejnojmenného výčtového typu – *Policko*.

Třída obsahuje především metody, které se týkají změny, či prohození stavu jednoho, či dvou políček. Jelikož je hrací plocha nelineární, třída tak obsahuje pomocné metody, které dokáží rozlišit, zdali jsou dvě zadaná políčka svými sousedy.



Obrázek 24: Diagram tříd



Obrázek 25: Třída Souradnice

Třída je také nositelem dvou informací pro dále zmíněnou vykreslovací vrstvu aplikace. První tato informace udává, jakou figurku hráč označil ke svému budoucímu tahu. Druhou je seznam políček, přes které se pohybovala figurka počítačového hráče ve svém minulém tahu. Tyto dvě informace nejsou nikterak důležité pro chod, či výpočet cest v programu, pomáhají však lidskému hráči v přehlednosti a orientaci.

Z programového hlediska třída spolupracuje jednak s řídícím Controllerem, který ji zasílá požadavky na úpravu pozic herních figurek. Též se této třídy dotazuje na provedení možného tahu, např. skoku. Nutná je také spolupráce s níže uvedenými controllery pro výpočet počítačového tahu. Poslední vazba je na samotnou zobrazovací vrstvu aplikace. Ta z této třídy čerpá podklady o jednotlivých políčkách, ale také o tom, zdali uživatel vybral některou svoji figurku pro následující tah. Poslední dotaz pro vykreslování je ten, který vyobrazuje informaci o tom, kudy počítač provedl svůj tah.

Datová vrstva také obsahuje třídy, které uchovávají data pro výpočet tahu počítačového hráče. Základním herním prvkem hry Čínská dáma jsou kameny, známější spíše pod pojmem figurky.

Pro spočítání správného tahu tak aplikace musí obsahovat jejich vhodnou reprezentaci, v tomto případě také pod stejným názvem, *Figurka*. Pro přehlednost jejich uchování je využita třída vystavěná nad lineárním seznamem, pojmenovaná příhodně *SeznamFigurek*.

Jelikož počítačový hráč může ve svém tahu použít pouze své figurky a se soupeřovými nesmí pohnout, tak seznam obsahuje pouze jeho vlastní.

Informaci o umístění figurek lze také vyčíst z již zmíněného *HerníhoPole*. Při každém započatém tahu by však algoritmus musel nejprve nalézt všechny počítačem řízené figurky uprostřed celé plochy, což by negativně ovlivňovalo rychlost výpočtu.

Při výpočtu jednotlivých řešení je potřebné nejprve uložit ke každé figurce informaci o tom, na které pozice se může v daném kole přesunout, a to jak posunutím na sousední pole, tak skokem, či několika skoky v řadě (zřetězení skoků). Seznam souřadnic (políček), přes která se figurka může přemístit, je uložen vždy v instanci třídy *SeznamTahu*. Jedná se o datovou strukturu navrženou nad lineárním seznamem. Obdobu této třídy také obsahuje výše zmíněné *HerniPole*.

Jelikož se figurka ve valné většině případů může pohybovat více směry a skončit na různých políčkách, aplikace již od začátku obsahuje třídu *SeznamReseni*. Jedná se opět o čistě datovou třídu vytvořenou nad lineárním seznamem, která obsahuje jednotlivé *SeznamyTahu*, které může figurka v daném kole provést. Právě tato třída je hlavním nosičem informací, které ohodnocuje a ze kterých si vybírá nejvhodnější řešení za pomocí metod k tomu určenými.

Všechny třídy obsahující seznamy implementují také třídu *Iterable*. Díky tomu lze jednotlivé seznamy procházet za pomoci cyklu zvaného for-each.[16]

Obrázek 26: Třída HerniPole s vazbami na ostatní třídy

10.3 Zobrazovací vrstva aplikace

Tato vrstva obsahuje grafickou část aplikace a celé uživatelského rozhraní. Vše, co je vyobrazeno koncovému uživateli, je obsluhováno právě těmito třídami. Jedná se o dvě třídy:

- *HerniPanel,*
- *MainFrame.*

MainFrame je třída odvozená od *JFrame*, základního stavebního prvku jakékoliv grafického programu. Jeho součástí je tlačítkové menu a herní plocha (reprezentovaná pomocí třídy *HerniPanel*). Třída úzce spolupracuje jednak s touto plochou, ale také s řídícím controllerem aplikace. Tomu předává informace o uživatelské interakci. Ať už se jedná o označování figurky k následnému tahu, či informaci o tom, že hráč chce ukončit svůj tah. Samozřejmostí je tlačítkové menu, ve kterém si uživatel může hru restartovat (začít novou hru), či ukončit.[14] [13]

V třídě *HerniPanel*, odvozená od *JPanel*, je pak řešeno vykreslování herní plochy, tedy šesticípé hvězdy. Vykreslování je řešeno přetížením metody *Paint*. Třída také musí obsahovat posluchače (listenery) pro zaznamenání kliknutí myši uživatele. Zachycené události předává své nadřízené třídě, *MainFrame*, která v případě potřeby informuje řídící controller aplikace.[19]

Zobrazovací vrstva aplikace tedy komunikuje převážně s řídícím controllerem, kterému předává informace o jednotlivých uživatelských akcích. Třída také musí komunikovat s modelem, se samotným herním polem (třída *HerniPole*), které má uživateli vyobrazit.

10.4 Řídící vrstva aplikace

Zde se nachází třídy obstarávající funkčnost programu. Bez této vrstvy by hra byla pouze statická a odrážela pouze jeden stav. Jakákoliv změna datového stavu aplikace je tedy vyvolána požadavkem těchto tříd. Jednotlivé controllery tedy slouží pro obsluhu uživatelské interakce, pro výpočty všech možných tahů pro počítač, následný výběr nejvýhodnějšího tahu a nakonec i jeho samotnou realizaci. Řídící vrstva obsahuje celkem tři controllery:

- *MainController,*
- *MetodySkoku,*
- *TahPocitace.*

MainController obsahuje funkce, které kontrolují interakci hráče. Zároveň je pomyslným propojením všech funkčních celků celého programu. Rozhoduje nejen o tom, zdali na základě kliknutí myši jde o označení figurky pro tah, o posun figurky, či o provedení skoku, či o konec hráčova tahu. Musí také umět rozhodnout, zdali hráčem požadovaný tah je vůbec možný. Úkolem třídy také je ověřovat, zdali se model (herní plocha) po provedení tahu nedostane do stavu, kdy

může být jeden z hráčů prohlášen za vítěze. Vzhledem k náročnosti a obsáhlosti úloh deleguje metody pro výpočet do jiných dvou controllerů, a to do instancí tříd *MetodySkoku* a *TahPocitace*.

MetodySkoku je třída starající se o vyhledávání všech možných tahů pro počítačového hráče, především pak skoků. Nachází se zde rekurzivní funkce pro vyhledávání všech možných tahů v daném kole. Dále ve třídě máme funkce, které ukládají vybrané vhodné tahy do seznamu.

TahPocitace pak slouží k výběru určitého tahu, který má počítač provést. Zpracovává se zde tedy seznam vytvořený třídou *MetodySkoku*. O správném výběru tahu pojednává kapitola Optimalizace výběru tahů.

Některé metody tříd, které jsou zde zmíněny, budou více popsány v kapitole věnované popisu metod.

11 Popis vybraných metod

V herním poli se nachází velmi důležitá metoda, která slouží k získávání souřadnic sousedního políčka. Jako parametry se této metodě pošlou souřadnice políčka, u kterého je potřeba znát sousední a směr ve kterém se má hledat. Vždy probíhá kontrola, zda se sousední políčko nehledá mimo rozměry pole. Když takové políčko existuje, vrací se jeho souřadnice.

```
public Souradnice getSouradniceSousednihoPole(Souradnice souradnice, Smer smer)
   {

     int radek = souradnice.getRadek();
     int sloupec = souradnice.getSloupec();

     switch(smer){
        case levoNahoru: if((radek-1) > -1 && (sloupec-1) > -1){
                     return new Souradnice(radek-1,sloupec-1);
                     }break;
        case pravoNahoru: if((radek-1)> -1 && (sloupec+1) < 25){
                     return new Souradnice(radek-1, sloupec+1);
                     }break;
        case levoDolu:  if((radek+1) < 17 && (sloupec-1) > -1){
                     return new Souradnice (radek+1, sloupec-1);
                     }break;
        case pravoDolu: if((radek+1) < 17 && (sloupec+1) < 25){
                     return new Souradnice (radek+1, sloupec+1);
                     }break;
        case levo:      if((sloupec-2) > -1){
                     return new Souradnice (radek, sloupec-2);
                     }break;
        case pravo:     if((sloupec+2) < 25){
                     return new Souradnice (radek, sloupec+2);
                     }break;
     }
     return null;
  }
```

Výpis 1: Metoda pro zjištění sousedů

Další metoda, která zde bude popsána, slouží k vyhledávání možných tahů, v tomto případě skoků. Nazývá se*vytvorSeznamySkoku*. Jedná se o rekurzivní metodu. Jako parametry se jí posílají souřadnice. Ze zadaných souřadnic metoda rekurzivně zkouší provést další skok. Pokud

takovýto skok existuje, metoda opět zavolá sama sebe a pokouší se najít další. Tímto stylem metoda prochází vždy všechny směry skoků a následně i posunů. Jednotlivé úspěšně nalezené tahy se ukládají do seznamu. Ve chvíli, kdy už nemá možnost skákat dále v daném směru, vrací se o políčko zpět (to zajišťuje poslední příkaz této metody).

```
  private void vytvorSeznamySkoku(Souradnice souradnice){

   // Zkouska posunuti figurky do smeru
   Souradnice nove;
   // Pokud muzu tahnout vlevo dolu -> zkusit dalsi skok
   nove = posun_o2(Smer.levoDolu, souradnice);
   if (nove != null) {vytvorSeznamySkoku(nove);}

   // Pokud muzu tahnout vpravo dolu -> zkusit dalsi skok
   nove = posun_o2(Smer.pravoDolu, souradnice);
   if (nove != null) {vytvorSeznamySkoku(nove);}

   // Pokud muzu tahnout levo dolu -> zkusit dalsi skok
   nove = posun_o2(Smer.levo, souradnice);
   if (nove != null) {vytvorSeznamySkoku(nove);}

   // Pokud muzu tahnout pravo dolu -> zkusit dalsi skok
   nove = posun_o2(Smer.pravo, souradnice);
   if (nove != null) {vytvorSeznamySkoku(nove);}

   // Pokud muzu tahnout vlevo nahoru -> zkusit dalsi skok
   nove = posun_o2(Smer.levoNahoru, souradnice);
   if (nove != null) { vytvorSeznamySkoku(nove);}

   // Pokud muzu tahnout vpravo nahoru -> zkusit dalsi skok
   nove = posun_o2(Smer.pravoNahoru, souradnice);
   if (nove != null) { vytvorSeznamySkoku(nove);}

   // Pokud uz neni kam tahnout -> vratime se zpet.
   seznamTahu.smazPosledniPrvek();
}
```

Výpis 2: Rekurze pro vyhledávání skoků

V předchozí metodě je volána metoda *posun_o2*. Té se jako parametry posílají souřadnice a směr. Nejprve se provádí kontrola, zda je možno skok provést. V případě, že to nelze, vrací

tato metoda null. Pokud ano, provádí se kontrola, zda již na toto políčko nebylo figurkou taženo. Touto kontrolou se zamezí stavu, kdy by se rekurze zacyklila na přeskakování jedné figurky donekonečna. Pokud se jedná o nový tah, přidá se do seznamu.

```
private Souradnice posun_o2(Smer smer, Souradnice souradnice ) {

   // Lze provest skok zadanym smerem?
   Souradnice nove = herniPole.lzeProvestSkok(smer, souradnice);

   // Pokud ne, vrat null.
   if (nove == null) {return null;}

   // Pokud lze, pridej nove reseni.
   if (test_proti_zacykleni(nove) ) {
      seznamTahu.pridejTah(nove);
      figurka.pridejReseni(seznamTahu);
      return nove;
   }
   return null;
}
```

Výpis 3: Metoda pro přidání možného skoku do seznamu

Následující metoda *ohodnot_reseni* slouží jak již název napovídá pro ohodnocování jednotlivých řešení. Jako parametr se jí zasílá *SeznamTahu*. Pro to aby mohl být tah ohodnocen, je nutné, aby měl počáteční i koncové souřadnice (pokud tyto souřadnice chybí, řešení neexistuje). Základní hodnocení vzniká z rozdílu souřadnic řádků. Tato metoda je značně modulární, může obsahovat mnoho různých podmínek, které upravují výsledné hodnocení. V metodě jsou postihovány například tahy vzdálené o více než tři indexy od středového sloupce a drobný postih dostávají také figurky, které se již nacházejí v domečku. Naopak se přičítá bonus pro figurky, které zůstaly příliš pozadu na herní ploše.

```
private double ohodnot_reseni(SeznamTahu reseni) {
   Souradnice startovni = reseni.getPrvniIndex();
   Souradnice cilove = reseni.getPosledniIndex();

   // Neexistujici reseni pokud tah nema start nebo cil...
   if (startovni == null || cilove == null) {
      return this.MIN_HODNOTA; // -1000
   }

   // Pocatecni hodnoceni.
```

```
    double hodnoceni = cilove.getRadek() - startovni.getRadek();

    // Penalizace za tahy do stran od stredu.
    double vzdalenost_od_stredu;
    if (reseni.getPosledniIndex().getRadek() % 2 == 0) {
       vzdalenost_od_stredu = Math.abs(12 - cilove.getSloupec() );
    }else {
       vzdalenost_od_stredu = Math.min(
       Math.abs(11 - cilove.getSloupec() ),
       Math.abs(13 - cilove.getSloupec() )
       );
    }

    // Pokud tah vedle daleko od stredu, penalizuj mu ohodnoceni.
    if (vzdalenost_od_stredu > 3) {
       hodnoceni -= (vzdalenost_od_stredu / 2.0);
    }

    // Bonusove hodnoceni pro figurky v horni casti herni plochy.
    if (startovni.getRadek() < 7){
       hodnoceni += 0.5;
    }

    // Figurka je hodne hluboko ve svem domecku.
    if (startovni.getRadek() < 3){
       hodnoceni += 1;
    }

    // Postih pro tahy s figurkou v cilovem domecku.
    if (startovni.getRadek() > 12) {
       hodnoceni -= 0.5;
    }

    return hodnoceni;
}
```

Výpis 4: Metoda pro ohodnocování tahů

Dále zde budou uvedeny metody z tříd tvořících seznamy. Zdrojový kód seznamu figurek počítače je zobrazen níže. Každá figurka má své souřadnice, v tomto seznamu jsou to souřadnice,

na kterých jsou figurky rozestavěny na začátku hry.

```
public SeznamFigurek() {

   // Zadani pocatecnich figurek pocitace.
   seznamFigurek.add(new Figurka(0, 12));
   seznamFigurek.add(new Figurka(1, 11));
   seznamFigurek.add(new Figurka(1, 13));
   seznamFigurek.add(new Figurka(2, 10));
   seznamFigurek.add(new Figurka(2, 12));
   seznamFigurek.add(new Figurka(2, 14));
   seznamFigurek.add(new Figurka(3, 9));
   seznamFigurek.add(new Figurka(3, 11));
   seznamFigurek.add(new Figurka(3, 13));
   seznamFigurek.add(new Figurka(3, 15));
}
```

Výpis 5: Seznam figurek

Poslední metoda, která zde bude uvedena je *Iterator<Figurka> iterator*. Ta obsahuje metodu pro kontrolu, že v seznamu ještě existuje další prvek *hasNext()*. Dále pak metodu pro získání následujícího prvku *next()*. Další pak pro možnost odstranění prvku ze seznamu *remove()*. Vzhledem k tomu, že seznam budeme využívat pouze v režimu pro čtení, metodu tak implementovat nebudeme. [17]

```
@Override
public Iterator<Figurka> iterator() {
   Iterator<Figurka> it = new Iterator<Figurka>() {

      private int index = 0;

      @Override
      public boolean hasNext() {
         return index < getPocetPrvku();
      }

      @Override
      public Figurka next() {
         return vratFigurku(index++);
      }

      @Override
```

```
      public void remove() {
         throw new UnsupportedOperationException();
      }
   };
return it;
}
```

Výpis 6: Iterator

12 Závěr

Úkolem této práce bylo seznámení čtenáře s hrou zvanou Čínská dáma. Úvodní kapitoly práce se věnovaly popisu hry a její historii. Následující kapitoly popisovaly několik variant této deskové hry, které se od sebe lišily různými pravidly. Při vytváření aplikace v této práci byla zvolena základní varianta, při které se mohou figurky pohybovat všemi směry, ale pohyb typu skok lze uplatnit pouze přes sousedící figurky. Hra je od začátku vytvářena jen pro dva hráče, a to pro souboj hráče proti počítači.

Před začátkem vlastní implementace bylo nutné provést analýzu této hry, a to zejména se zaměřením na tvorbu umělé inteligence, která bude reprezentovat počítačového hráče. Po provedení analýzy byl pro implementaci s ohledem na objektově orientovanou architekturu zvolen programovací jazyk Java. Program byl nejdříve vyvíjen jako konzolová aplikace. Až po ověření funkčnosti základních prvků byl převeden do grafické podoby. Aplikace se snaží jít v duchu návrhového vzoru MVC, který je v teoretické části také krátce popsán. Tato část také obsahuje popis principu všech datových struktur, o které se vytvářená aplikace postupně rozšiřovala. Zejména je kladen důraz na popsání struktury herního pole, základního stavebního prvku aplikace. Krom uvedení datových struktur práce také obsahuje popis tvorby jak grafického rozhraní, tak i řídícího controlleru.

Po vytvoření prvního prototypu práce, se vytvořila ukázka zvoleného řešení se zaměřením na výběr tahů, které algoritmus počítačového hráče volil. Pilotní verze nebyla nijak optimalizovaná a tak algoritmus často volil tahy, které se na první pohled zdají jako nejlepší. V podání několika tahů se jimi však dostal do nevýhodné pozice, kdy většina jeho figurek skončila na osamocených částech herního pole. Existovaly i případy, kdy počítač s některými figurkami poprvé zahrál až skoro ke konci partie. Metoda, která algoritmu pomáhá vyhodnocovat kvalitu jednotlivých řešení tak v průběhu tvorby práce doznala značných změn. Po jejich uvedení se tahy počítače již zdají dostačující. Práce také obsahuje srovnání této hodnotící metody s původní, neoptimalizovanou. Aplikace však ani po optimalizaci neobsahuje predikci, strojové učení, ani jinou pokročilou metodu výběru tahů. Hodnotící funkce je však dokáže do jisté míry nahradit.

Pro otestování kvality byla provedena a popsána simulace souboje dvou různých umělých inteligencí. V těchto partiích proti sobě stála námi vytvořená umělá inteligence proti aplikaci Chinese Checkers od autora Zheng Yi, volně dostupné z obchodu Google Play. [10] V tomto souboji náš algoritmus obstál, dokázal vyhrát 4:1 na zápasy.

Možnosti rozšíření aplikace mohou spočívat v úpravě grafického rozhraní, které neumožňuje volit barvy, ani pozice figurek jednotlivých kamenů. Hra by se také dala rozšířit o hru pro více hráčů, ať už lidských, či počítačových řízených umělou inteligencí. Aby však umělá inteligence dosahovala špičkových kvalit, musela by se rozšířit o jednu z metod predikce tahů, či případně využit jinou pokročilou techniku na bázi strojového učení. Jednou z možností, jak aplikaci udělat dynamičtější by bylo i zavedení možnosti úpravy herních pravidel. Příkladem může být možnost využít delší skoky, tak jak byly popsány v teoretické části věnující se různým variantám hry.

Literatura

[1] ZAPLETAL, Miloš. *Velká kniha deskových her*. 1. Brno: Mladá fronta, 1991. ISBN 80-204-0188-1.

[2] ZAPLETAL, Miloš. *Velká encyklopedie her 2: Hry v klubovně*. 1. Praha: Olympia, 1986. ISBN 27-053-86.

[3] H. ROHWEDDER, Lars. Čínská dáma. In: *Wikipedie* [online]. San Francisco, 2007 [cit. 2019-03-12]. Dostupné z: https://commons.wikimedia.org/wiki/File:Sternhalma.svg

[4] Čínská dáma. *Čínská dáma* [online]. 2009 [cit. 2019-03-12]. Dostupné z: http://www.deskovehry.info/pravidla/cin-dama.htm

[5] Chinese checkers rules. *Playluggage* [online]. c2011 [cit. 2019-04-20]. Dostupné z: http://www.playluggage.com/Chinese-checkers-rules

[6] ROGNLIE, Richard. ChineseCheckers. *Gamerz.NET Enterprises* [online]. 2015 [cit. 2019-03-12]. Dostupné z: http://www.gamerz.net/ pbmserv/chinesecheckers.html

[7] WHITEHILL, Bruce. Chinese Checkers. *The Big Game Hunter* [online]. c2019 [cit. 2019-03-12]. Dostupné z: http://thebiggamehunter.com/games-one-by-one/chinese-checkers/

[8] Chinese Checkers. *Sweetoothdesign* [online]. Sweetooth Design, c2014-2018 [cit. 2019-03-12]. Dostupné z: https://www.sweetoothdesign.com/games-chinese-checker

[9] Hra Čínská dáma. *Free Online Fun Arcade Games - Play Fun Arcade Games* [online]. c2017 [cit. 2019-04-16]. Dostupné z: http://www.bigmoneyarcade.com/?action=playgame&gameid=47

[10] ZHENG, Yi. Chinese Checkers. *Google Play* [online]. Dublin: Google Commerce Limited, 2008, 2018 [cit. 2019-04-20]. Dostupné z: https://play.google.com/store/apps/details?id=com.xunyougame.checkers&hl=en_US

[11] PEKAŘ, Lukáš. Model-View-Controller. *Bonsai Development* [online]. 2018, 4.9.2018 [cit. 2019-03-20]. Dostupné z: https://bonsai-development.cz/clanek/co-je-to-model-view-controller

[12] BERNARD, Borek. Úvod do architektury MVC. *Zdroják - o tvorbě webových stránek a aplikací* [online]. Praha: Devel.cz Lab, c2019, 7.5.2009 [cit. 2019-04-22]. Dostupné z: https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[13] JAISWAL, Sonoo. Java JPanel. *Tutorials-Javatpoint* [online]. JavaTpoint, c2011-2018 [cit. 2019-04-20]. Dostupné z: https://www.javatpoint.com/java-jpanel

[14] JAISWAL, Sonoo. Java JFrame. *Tutorials-Javatpoint* [online]. JavaTpoint, c2011-2018 [cit. 2019-04-20]. Dostupné z: https://www.javatpoint.com/java-jframe

[15] ArrayList (Java Platform SE 8). *Oracle Help Center* [online]. Redwood Shores: Oracle Corporation, ©1993-2019 [cit. 2019-04-20]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

[16] JENKOV, Jakob. Java Iterable. *Jenkov.com* [online]. Denmark: Jenkov Aps, c2019, 2019 [cit. 2019-04-23]. Dostupné z: http://tutorials.jenkov.com/java-collections/iterable.html

[17] PAWAR, Sejal. Implementing iterator. *GeeksforGeeks* [online]. Noida [cit. 2019-04-20]. Dostupné z: https://www.geeksforgeeks.org/java-implementing-iterator-and-iterable-interface/

[18] KUMAR, Abhishek. Equals() and hashCode(). *GeeksforGeeks* [online]. Noida, 2018 [cit. 2019-04-20]. Dostupné z: https://www.geeksforgeeks.org/equals-hashcode-methods-java/

[19] Writing Event Listeners. *Oracle* [online]. Redwood Shores: Oracle Corporation, c1995-2017 [cit. 2019-04-20]. Dostupné z: https://docs.oracle.com/javase/tutorial/uiswing/events/mouselistener.html

[20] ČÁPKA, David. UML - Class diagram. *Itnetwork.cz* [online]. c2019, 2013 [cit. 2019-04-24]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-class-diagram-tridni-model

A Příloha v IS EDISON a na přiloženém CD

Přílohou je CD, které obsahuje projekt vytvořený v rámci této bakalářské práce. Jeho součástí jsou zdrojové kódy i zkompilovaná aplikace. Součástí je také dokumentace ve formátu Javadoc. Obsah CD byl také odevzdán na portálu IS EDISON.